L E S Y S T E M E
V L I S P 1 6

DECEMBRE 1978

Patrick GREUSSAY

Université Paris-8-Vincennes

et

LITP

TABLE DES MATIERES

CHAPITRE 1

INTRODUCTION

Le système VLISP 16, implémenté sur ordinateur SOLAR 16, est une version destinée à l'Ecole Polytechnique du système VLISP de l'Université PARIS-8-VINCENNES.

Le langage et le système sont destinés à l'enseignement et à la recherche en programmation expérimentale, en intelligence artificielle, et en informatique musicale.

VLISP 16 est une des plus récentes versions de VLISP, déjà implémenté sur une grande variété d'ordinateurs, très anciens (CAE 510, CAB 500), plutôt récents (T1600, SOLAR), grands ordinateurs (DEC PDP 10), et microprocesseurs (Zilog Z80, INTEL 8080).

VLISP 16 peut, dans son état actuel, être utilisé sous système RBOS/D en mode maître et mono-utilisateur, ainsi qu'en temps partagé sous système TSF multi-langages. VLISP est, dans ces deux modes, totalement interactif, et vient ainsi compléter sous TSF l'ensemble des langages de programmation disponibles a l'Ecole.

Ce manuel de référence livre une description informelle du langage VLISP 16 et de son système, chaque construction décrite étant illustrée par une grande variété d'exemples ponctuels.

Le dernier chapitre donnera en revanche un certain nombre de programmes complets d'une certaine ampleur, supposés représentatifs des techniques et intentions mises en jeu dans les programmes LISP contemporains. Ces programmes viennent ainsi compléter l'énumeration des constructions du langage en les rassemblant par des intentions et des styles très variés de programmation. L'auteur engage vivement le lecteur à les étudier avec soin, à les implémenter et les développer.

Ce manuel n'est pas réellement destiné aux débutants, mais aux utilisateurs ayant déjà une certaine expérience de LISP. Le lecteur ne doit donc pas espérer une introduction progressive et ordonnée aux constructions du langage : les références avant y seront plus souvent la régle que l'exception dans les exemples. C'est ainsi.

Le manuel décrit en principe tous les aspects courants du système, mais n'engage l'auteur que sur les principes de base. Le manuel sera sujet à des remaniements, au fur et à mesure de l'évolution normale et souhaitée du système et de ses utilisateurs.

Si vous détectez une situation très anomalique ou une erreur manifeste du système, rassemblez le plus grand nombre d'informations sur son état courant, et téléphonez-moi toute affaire cessante à :

Patrick GREUSSAY
Département d'Informatique
Université Paris-8-Vincennes
8216364

Je crois à propos de remercier J. CHAILLOUX, G. ENGLERT, H. WERTZ, J.C. HALGAND, P.L. NEUMAN, D. GOOSSENS, J. POINDRON, J.E. SCHOETTL, J.P. BOUDIER, S. CHARALAMBIDES, J.F. PERROT, B. ROBINET, H. HUITRIC, M. NAHAS, J. VIGNOLLES, J.L. DURIEUX, G. PAUL, A. CATTENAT, B. MEYER, J. ALLEN, J. LAUBSCH, R. WEYRAUCH qui ont influencé, par leurs excellentes suggestions la conception du systeme dont ce manuel est le manuel.

L'arrivée de VLISP 16 a l'Ecole a été rendue possible grâce a J.P. CRESTIN, M. NIVAT, et J.P. JOUANNAUD.

J. ZEITOUN et A. BUIS ont permis et encouragé, a l'Institut de l'Environnement, la construction d'une version préliminaire de VLISP 16.

Ce manuel a été édité par l'auteur a l'IRCAM sur ordinateur PDP 10, grâce a la bienveillance et l'intérêt de G. BENNETT et J.C. RISSET.

L'édition a été réalisée avec le programme RF de J.L. RICHER, et l'impression du fichier édité a été réalisée sur l'imprimante électrostatique VERSATEC grâce a la compétence de R. BARA.

1.1 UNE SESSION INTERACTIVE AVEC VLISP 16.

Rien de meilleur, pour demarrer, que quelques exemples complets. En voici, sous la forme de sessions illustrant pour des travaux simples, quelques aventures avec VLISP 16. Je conseille fortement au nouveau venu sur le systeme de les rejouer pour son compte.

```
>RUN VLISP-:S,,'2FA8         ~Pour commencer sous TSF.
>LISP                        ~Pour demarrer a froid.
 VLISP IS WINNING AGAIN      ~Repond VLISP.
?()                          ~Pour se faire les doigts.
 NIL                         ~VLISP se fait la voix.
?12                          ~Un nombre
 12                          ~s'auto-denote.
?IT                          ~La variable IT est liee a la
 12                          ~derniere reponse au TOP-LEVEL.
?(+ IT IT)
 24                          ~La preuve.
?IT                          ~Pour le Sceptique
 24                          ~de l'Ecriture.
?                            ~RETURN pour aerer.
?
?(SETQ A '((A + B) * (A - B)))
 ((A + B) * (A - B))         ~La valeur de A.
?
?(DE POL (E)                 ~Definissons une fonction pour
?   (IF (ATOM E) E           ~poloniser prefixe une expression 2-aire.
?       [(CADR E) (POL (CAR E)) (POL (CADDR E))]))
 POL                         ~Enregistree OK.
?
?(POL A)                     ~On appelle la fonction POL
 (* (+ A B) (- A B))         ~qui ramene le resultat idoine.
?
?(POL '((B ↑ 2) - (4 * (A * C))))
 (- (↑ B 2) (* 4 (* A C)))
?
?'MERCA\I                     ~Je voulais dire I et non A.
 MERCI                        ~Le caractere d'annulation a marche.
?                             ~Je tape ESCAPE.
>                             ~Nous revoila sous TSF.
>CLISP                        ~On redemarre VLISP a chaud cette fois.
?
?(PRETTY POL)                 ~POL vit toujours. Je le pretty-printe.
                              ~On attend quelques instants.
                              ~Et voici que s'imprime
(DE POL (E)
   (IF (ATOM E) E (LIST (CADR E) (POL (CAR E)) (POL (CADDR E)))))
 PRETTY                       ~Le pretty-print est charge.
?
```

```
?(DE FOO (X)                    ~Definissons la reine des fonctions.
?    (IFE (ZERZOP\              ~Horrible. Annulons toute la ligne.
?    (IF (ZEROP X) 1 (TIMES X (FOO (1- X)))))
 FOO
?(FOO 7)                        ~Essayons.
 5040                           ~OK.
?
?(TRACE FOO)                    ~Je trace FOO.
                                ~TRACE s'auto-loade.
 (FOO)                          ~OK. FOO est tracee.
?(FOO 7)                        ~Lancons un appel avec trace.
------> 1    FOO        (7)     ~La ------> montre que ca plonge.
------> 2    FOO        (6)     ~Le numero d'appel suit.
------> 3    FOO        (5)     ~Le nom de la fonction appelee suit.
------> 4    FOO        (4)     ~Puis la liste des valeurs d'arguments.
------> 5    FOO        (3)
------> 6    FOO        (2)
------> 7    FOO        (1)
------> 8    FOO        (0)
<------ 8    FOO        1       ~La <------ dit qu'on remonte.
<------ 7    FOO        1       ~Suit le numero d'appel correspondant.
<------ 6    FOO        2       ~Puis le nom de la fonction dont on ramene
<------ 5    FOO        6       ~le resultat qui suit.
<------ 4    FOO        24
<------ 3    FOO        120
<------ 2    FOO        720
<------ 1    FOO        5040
5040                            ~Le resultat final.
?
?(UNTRACE FOO)                  ~Pour detracer FOO.
 (FOO)
?(FOO 7)
 5040                           ~Comme si de rien n'etait.
?                               ~Je tape ESCAPE.
>                               ~Retour a TSF.
```

1.2 UNE AUTRE SESSION AVEC VLISP ET EDIT16.

```
>CALL EDIT16                  ~Appelons l'editeur.
>EDIT SHUFFL-PG               ~Creons le fichier du meme nom.
NEW FILE                      ~Il est nouveau
INPUT:                        ~chante EDIT16.
(DE SHUF (FX GY) (COND        ~Et je tape une fonction qui va me
  ((NULL FX) [GY])            ~calculer le shuffle-product de 2 suites
  ((NULL GY) [FX])            ~representees par les listes FX et FY.
  (T (APPEND (DCONS (CAR FX) (SHUF (CDR FX) GY))
             (DCONS (CAR GY) (SHUF (CDR GY) FX))))))
;;                            ~Un commentaire vide. C'est la faute a
;;                            ~ce perdant d'EDIT16 qui crois qu'une
;;                            ~ligne vide signifie la fin de l'entree.
                              ~Pour avoir une ligne blanche, je tape
                              ~un blanc, puis RETURN. Dur.
(DE DCONS (X L)
  (MAPCAR L (LAMBDA (Y) (CONS X Y))))

; (DCONS x (e1 e2 ... eN)) ☞ ((x . e1) (x . e2) ... (x . eN)) ;

                              ~J'ai commente DCONS. Commentaire plein
                              ~entre deux point-virgules.
(PRINT 'SHUFFLPG 'IS 'WINNING 'AGAIN)
                              ~Pour que le chargement soit apparent.
(INPUT)                       ~Pour terminer le fichier.
                              ~Ligne vide. Fin d'entree.

                              ~Un fichier de fonctions qui ne
                              ~se termine pas par (INPUT)
                              ~TUE l'imprudent
                              ~ou le negligent.

F                             ~Je me casse d'EDIT16 en sauvegarde.
>                             ~Good old TSF is back again.
>RUN VLISP-:S,,'2FA8          ~On appelle un VLISP
>LISP                         ~tout neuf.
 VLISP IS WINNING AGAIN       ~Certes.
?(LIBRARY SHUFFLPG)           ~Je charge mon fichier (SANS le tiret !!!).
 SHUFFLPG IS WINNING AGAIN
 (SHUFFLPG)                   ~Le voila charge.
?
?(MAPC (SHUF '(A B) '(C D)) 'PRINT)
 (A B C D)                    ~Les suites shufflees sont imprimees.
 (A C D B)
 (A C B D)
 (C D A B)
 (C A B D)
 (C A D B)
 NIL                          ~NIL est la valeur d'un appel de MAPC.
?
?                             ~Vu ?. ESCAPE
>                             ~TSF
```

1.3 ENCORE UNE SESSION AVEC VLISP ET L'EDITEUR EF.

```
>RUN VLISP-:S,,'2FA8     ~On lance VLISP
>LISP                    ~a froid.
 VLISP IS WINNING AGAIN ~Comme toujours.
?
?(DE FOO (X Y)           ~Et on tape une version un peu
?    (FI (ATOM X)        ~fausse d'une fonction de copie de liste.
?        (CONS (FOO (CAR X)) (FOO CDR X)))
 FOO
?                        ~Pas fameux.
                         ~Faute de disposer du programme
                         ~PHENARETE d'Harald WERTZ, editons-la,
?(LIBRARY DEBUGG:L)      ~en chargeant DEBUGG-:L, qui contient
                         ~l'editeur EF, ADVISE et BREAK.
 (DEBUGG:L)              ~C'est chose faite.
?
?(EF FOO)                ~Pour que EF connaisse FOO.
 (LAMBDA (* *) (* * *)) ~EF n'imprime qu'a 1 niveau.
 *                       ~Signale la fin d'une action de EF.

?(P 2)                   ~Imprimons a 2 niveaux.
 (LAMBDA (X Y) (FI (* *) (* * *)))
 *
?(P 3)                   ~A trois.
 (LAMBDA (X Y) (FI (ATOM X) (CONS (* *) (* * *))))
 *                       ~Vu ?

?(FK X)                  ~Cherchons de haut en bas, et de gauche
                         ~a droite un segment de liste debutant par
                         ~la konstante X (Find Konstante).
 (X Y)                   ~Le voila. C'est la liste d'arguments.
 *
?(DL)                    ~Enlevons le dernier element (Delete Last).
 (X)
 *
?(EF FOO)                ~On reentre dans FOO pour avoir une vue
 (LAMBDA (*) (* * *))    ~d'ensemble. Ya plus qu'un argument OK.
 *
?(FK FI)                 ~Itou pour la clonstante FI.
 (FI (* *) (* * *))
 *
?(D 1)                   ~Enlevons ce FI inepte (Delete <nb-elements>).
 ((* *) (* * *))         ~Voila.
 *
?(I IF)                  ~Et inserons IF en tete (Insert <e1> ... <eN>).
 (IF (* *) (* * *))      ~Bon.
 *
?(EF FOO)                ~On retourne au rez-de-chaussee ou
 (LAMBDA (*) (* * *))    ~on pointe sur LAMBDA.
 *
```

```
?(MV 3 UP 3)               ~On pointe sur le 3eme element, on grimpe un
 ((* * *))                 ~etage de liste, et on pointe sur le 3eme
 *                         ~element de l'etage.
                           ~i.e. (MoVe <s1> ... <sN>)
                           ~avec <sI> ::= <n> | UP .

?(P 3)                     ~Voyons ca.
 ((CONS (FOO (* *)) (FOO CDR X)))
 *
?(I X)                     ~On colle X en tete.
 (X (* * *))
 *
?(EF FOO)                  ~On redescend.
 (LAMBDA (*) (* * X *))
 *
?(FK CONS)                 ~On cherche le 1er CONS venu.
 (CONS (* *) (* * *))
 *
?(DL)                      ~On enleve le dernier element.
 (CONS (* *))
 *
?(IL (FOO (CDR X)))        ~Et on place a la fin : (FOO (CDR X)).
 (CONS (* *) (* *))        ~i.e. (Insert Last).
 *
?(EF FOO)                  ~En bas.
 (LAMBDA (*) (* * X  *))
 *
?(P 6)                     ~Voyons le resultat de l'edition.
 (LAMBDA (X) (IF (ATOM X) X (CONS (FOO (CAR X)) (FOO (CDR X))))))
 *                         ~Correct.

?                          ~A present, testons.
?(FOO '(E (PLURIBUS) UNUM))
 (E (PLURIBUS) UNUM)       ~Ca colle.
```

1.4 GENERALITES

Les exemples de session qui precedent sont supposes offrir aux lispiens deja avertis la possibilite de programmer des choses simples sans tenir compte outre mesure des idiosyncrasies du systeme VLISP 16. On trouvera dans ce paragraphe l'expose de quelques particularites superficielles de VLISP 16. L'auteur ne s'illusionne pas quant a l'interet de cet expose. Il n'en est pas moins indispensable. Une FORME reste a inventer pour les manuels de references : systematiser l'expose, montrer la necessite des constructions, demontrer quand il faut, organiser l'essaim d'exemples comme un discours suivi, seduire pour convaincre, marquer enfin a tout instant l'aspect profondement liberateur et creatif de la programmation dans un langage capable.

Le besoin d'une telle forme est aujourd'hui sensible, son apparition demeure encore a venir. Elle ne s'est pas produite pour ce manuel, qui est probablement un des derniers du genre, en ce qui me concerne. Ce genre ancien se reconnait a ce qu'il distingue malaisement l'essentiel du superficiel et melange toutes les notions dans un joyeux fatras qui, aujourd'hui apparait moins eclatant.

C'est donc sans embarras que le present paragraphe deroule ses classifications semblables a celles des bestiaires de BORGES. Que le lecteur se persuade cependant qu'aux yeux de l'auteur, son itineraire n'en reste pas moins irremediablement absent.

1.4.1 Comment Utiliser VLISP 16

A l'Ecole, le mode normal d'utilisation sera un cycle de va-et-vient entre VLISP et EDIT16. Comme dans la seconde des sessions donnees en exemples, je vous conseille de rassembler vos paquets de definitions dans un fichier (voir Chapitre 5) que vous constituerez sous EDIT16, puis de les lire en bloc grace a la fonction LIBRARY.

Ces paquets une fois lus, vous appelez en mode interactif les fonctions qui les constituent, vous en constatez le plus generalement l'incorrection, vous testez un peu grace aux excellents outils du Chapitre 8, apres quoi vous revenez a EDIT16 pour rendre vos corrections definitives.

Il va de soi que ces ensembles de definitions, affectations usw., seront rassembles par une INTENTION (un traitement d'arbre, de demonstration automatique, de dialogue en langue naturelle, de verification de preuve, de resolution de probleme robotique, de calcul symbolique, d'enumeration d'objets combinatoires, de construction d'interpretes et de traducteurs, en general quelque chose d'interessant).

Pour entrer en VLISP 16, a l'Ecole chantez :

```
>RUN VLISP-:S,,'2FA8
>LISP
```

Le ">" est le prompteur de TSF. VLISP se glorifie un tant soit peu, apres quoi le caractere "?" prompteur de VLISP apparait. Vous tapez alors vos demandes d'evaluation.

Cependant, il se pourrait qu'une evaluation aboutisse a un desastre (voyez les desastres au Chapitre 7), patent dans le cas d'une impression interminable, muet dans le cas d'une boucle infinie. Interrompez le mauvais calcul en frappant la touche "BREAK". Retour se fait sous TSF. Pour revenir a VLISP sans tout reinitialiser (et du coup perdre votre image-memoire, pleine d'informations utiles), chantez :

```
>CLISP
```

Reste qu'il se pourrait que meme a appuyer sur BREAK, il ne se passe rien. Vous etes un homme mort. Votre image-memoire sera irremediablement detruite par les plusieurs appels de BREAK dans ce cas necessaires au retour sous TSF.

Pour sortir de VLISP, enfoncez la touche ESC, qui vous emmene a TSF.

1.4.2 Les Annulations

Pour annuler les <n> derniers caracteres tapes sur la ligne, frappez <n> fois le caractere "\". Puis tapez vos caracteres de remplacement a la suite.

Pour annuler d'un seul coup toute une ligne, tapez "\" suivi IMMEDIATEMENT du RETURN. Puis retapez votre ligne tranquillement.

Quand on definit une fonction, et que plusieurs lignes sont deja ecrites, et qu'on n'est pas content, et qu'on veut tout annuler, pour recommencer du bon pied, on tape "..." i.e. quelques points de suite suivis de RETURN : ce qui provoque une erreur de lecture et l'annulation recherchee.

1.4.3 Le Mode Eval

L'interprete tourne en mode EVAL : il lit un atome ou un appel de fonction, l'evalue, imprime le resultat de l'evaluation, et recommence. A peu pres ceci :

```
(WHILE T (SETQ IT (PRINT (EVAL (READ)))))
```

ou encore

```
(LET ((IT 'IT)) (SELF (PRINT (EVAL (READ)))))
```

Dans ce qui suit, une telle boucle sera nommee boucle TOP-LEVEL.

La variable globale IT sera liee avec le resultat de la derniere evaluation au TOP-LEVEL.

EXEMPLE :

```
?2
 2
?(* IT IT)
 4
?(* IT IT)
 16
```

Le terminal indique qu'il attend quelque chose a lire, par l'impression de "?". On tape alors a la suite. Il est possible d'eviter l'impression au TOP-LEVEL (voir Chapitre 6).

Il est egalement possible comme je viens de le faire de se redefinir soi-meme sa propre boucle TOP-LEVEL.

1.4.4 Commentaires

Tout ce qui se trouve entre deux ";" sera totalement ignore du lecteur VLISP. Un commentaire peut se deployer sur plusieurs lignes.

HORRIBLE EXEMPLE :

```
?(; horrible commentaire inepte ;)
 NIL
?N; encore un ;I; horrible commentaire ;L
 NIL
?NI;
?   un dernier
?   horrible commentaire
?;L
 NIL
```

A ce propos, je me sers assez souvent d'un commentaire vide pour delimiter, dans la liste d'arguments formels d'une fonction, les arguments que j'utilise effectivement a l'appel, de ceux que je considere comme des variables de travail locales a la fonction. Artifice purement typographique. Voir pourquoi ca marche au Chapitre 2.

EXEMPLE :

```
(DE REVDROP (L ;; R)
  (IF L (REVDROP (CDR L) (CONS [(CAR L)] R))
      R))

(REVDROP '(A B C))  ☞  ((C) (B) (A))
```

1.4.5 Format Externe Des Atomes

Il peut s'agir :

- d'un nombre i.e. une suite de chiffres, eventuellement precedee de "-". Une suite de chiffres precedee par "+" n'est PAS consideree comme un nombre.
- d'un symbole i.e. un atome non-numerique. Toutes les combinaisons de caracteres sont acceptees, a l'exception des separateurs. Tout atome qui n'est pas un nombre sera considere comme un symbole. A la lecture d'un symbole, seuls les 8 premiers caracteres sont enregistres, tous les caracteres suivants sont ignores.

1.4.6 Separateurs

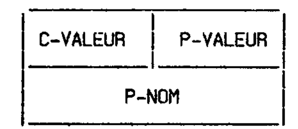
L'espace (" "), le point ("."), les parentheses ouvrantes ("("), et fermantes (")"), le quote ("'"), le carre ouvrant ("[") et fermant ("]"), sont des separateurs d'atomes, qui ne sauraient en comporter, sauf a les faire immediatement preceder du caractere slash ("/") qui masque alors leur qualite de separateurs.

Un macro-caractere (voir Chapitre 5) acquiert la qualite de separateur.

Le point-virgule de commentaire (";") n'est PAS un separateur.

1.4.7 Format Interne Des Atomes

Un atome-symbole est (tres approximativement) represente en memoire ainsi :



Les parties C-VALEUR et P-VALEUR contiennent des adresses de symboles, de nombres ou de listes.

La partie P-NOM contient la suite des caracteres qui rendent le symbole imprimable.

La partie C-VALEUR contiendra, a tout instant la valeur du symbole, considere comme variable, globale ou parametre de fonction.

Cette C-VALEUR sera explicitement accessible en tant que CAR de l'atome.

Les C-VALEURS des atomes definis par l'utilisateur sont initialisees a la valeur INDEFINI, ce qui provoquera l'erreur A8 (voir Chapitre 7), si l'evaluation en est tentee.

Les C-VALEURS des fonctions standard contiennent l'adresse d'un point d'entree dans le code-machine correspondant.

Certains atomes du systeme contiennent leur propre adresse en C-VALEUR, ces atomes speciaux nommes CONSTANTES, sont NIL, QUOTE, LAMBDA, EXPR, FEXPR, MACRO, T.

Nul besoin donc de les quoter.

EXEMPLE :

```
[NIL QUOTE LAMBDA EXPR FEXPR T]
           ->
(NIL QUOTE LAMBDA EXPR FEXPR T)
```

Il faut aussi savoir qu'en VLISP 16, tout appel avec, sauf NIL, une constante en position de fonction, vous ramene l'appel tel quel en valeur. Avec NIL, vous bouclez indefiniment.

EXEMPLES :

```
?(T FOO)  ☞  (T FOO)

?(LAMBDA (FOO) (BAR))
        ->
 (LAMBDA (FOO) (BAR))

?(NIL T)  ☞   boucle
```

Cette propriete s'etend a toute variable qui est liee a sa propre valeur.

La partie P-VALEUR contiendra la P-liste de l'atome. Elle sera consideree comme le CDR de l'atome.

La P-VALEUR des constantes est initialisee a NIL. Ainsi l'egalite suivante est satisfaite :

(CDR NIL) = NIL = (CAR NIL)

On notera que DE, DF et DM (voir le Chapitre 2) ne touchent pas a la C-VALEUR du nom de la fonction qu'elles permettent de definir. Il est donc loisible de redefinir une fonction standard.

EXEMPLE :

```
(DE CAR (L) (CDR L))           ~CAR se comporte comme CDR.
(CAR '(A B C))  ☞  (B C)       ~Comme on voit.

(REMPROP 'CAR EXPR)            ~Que CAR se comporte comme CAR.
(CAR '(A B C))  ☞  A           ~Voila.
```

1.4.8 S-Expressions Pointees

Elles sont acceptees par le systeme sous la forme la plus generale :

(<s-expr-pointee> . <s-expr-pointee>)

EXEMPLE :

((A . (B . C)) . (D E . (F G H . I)))

1.4.9 Les Nombres

Seulement les entiers. Et inclus dans l'intervalle [-32767,32767]. Les atomes numeriques ne comportent ni C-VALEUR, ni P-VALEUR, ni P-NOM. Ils sont stockes en zone liste dans un format special.

Des nombres hexadecimaux peuvent etre directement utilises. Ils ont la representation externe :

<c><c><c><c>HH

dans laquelle <c> est un chiffre hexadecimal ∈ [0,1, ... ,E,F].

1.4.10 Combien D'Objets ?

Par defaut 200 atomes-symboles definis par l'utilisateur, et 5200 doublets en zone liste (ces indications seront appelees a varier), il est tout a fait possible de regler ces nombres selon son desir (voir Chapitre 7).

Pour savoir combien de cellules de liste (doublets) sont disponibles, evaluez de temps en temps :

(CDR 'REM)

1.4.11 L'Evaluation Des Variables

VLISP 16, comme ses contemporains, ne represente PAS son environnement par une A-liste. A tout moment, la valeur des variables est accessibles en C-VALEUR. A l'entree d'un PROG ou d'une fonction dont la variable est argument formel, l'ancienne valeur de cette variable est empilee dans une pile de travail, et restituee a la sortie.

Ce mode d'acces est TRES efficient, repond a l'appellation de LIAISON-SUPERFICIELLE, et les versions de LISP qui ne le font pas sont perdantes. Mais tout se paye. Pas question de FUNARG en VLISP 16. Si vous voulez des fermetures programmez en SCHEME ou en PLASMA.

Voici cependant une facon possible de vous definir une fermeture de la forme :

(CLOSURE (<v1> ... <vN>) <λ-expression>)

Ou les <vI> sont les variables dont vous voulez voir la valeur courante FERMEE dans la <lambda-expression>. CLOSURE vous ramene une fonction telle que si vous l'appliquez, les valeurs des occurences libres des <vI> seront celles du moment de la construction de la fermeture (si vous avez compris cette phrase, nous vous offrons de venir a votre prix donner un cycle de conferences a Vincennes pour nous l'expliquer).

```
(DF CLOSURE (-L)
  [LAMBDA (CADADR -L) (CONS [LAMBDA (CAR -L) (CADDR (CADR -L)) ]
                            (MAPCAR (CAR -L)
                              (LAMBDA (-X) [QUOTE (EVAL -X) ]))) ])
```

On notera qu'en cas d'erreur et retour de ce fait au TOP-LEVEL, les variables conservent la valeur qu'elles avaient AU MOMENT de l'erreur : precieuses informations pour la mise au point.

1.4.12 VRAI Et FAUX

EN VLISP 16, la valeur logique FAUX se dit NIL, et la valeur logique VRAI est equivalente a non-NIL.

Ceci permet de faire ramener aux predicats des valeurs utiles.

Par exemple, (MEMQ x y) ramene, ou bien NIL, ou bien le CDR de y dont x est le premier element. De meme, la valeur de OR est la valeur de son premier argument vrai (i.e. non-NIL), et la valeur de AND est, ou bien NIL, ou bien la valeur de son dernier argument.

1.4.13 Les Interpretations Iteratives

Une particularite maitresse de VLISP, quelque soit l'implementation est d'interpreter iterativement les fausses recursivites. Iterativement est ici defini en termes de ressources : un appel de fonction est iteratif s'il ne demande pas plus de ressources que celles accordees a l'entree de la fonction. Les ressources en questions sont les tailles de piles, ainsi que le nombre de doublets.

C'est ainsi que dans :

```
(DE TOP-LEVEL (IT)
    (TOP-LEVEL (PRINT (EVAL (READ)))))
```

la suite des appels internes de TOP-LEVEL ne provoquera PAS un debordement de pile, et bouclera, comme il se doit pour une boucle-systeme, indefiniment.

Cette propriete se conserve, quelque soit le niveau d'imbrication des appels dits iteratifs dans les structures de controle mises en jeu dans le corps de fonction.

L'utilisation systematique de cette propriete induit un style tres souple et tres naturel de programmation recursive, bien eloigne des horribles structures de controle dites plates.

La theorie de la chose et sa methode d'implementation sont decrites dans un rapport de l'auteur publie au Laboratoire d'Informatique Theorique et Programmation. Demandez-le a la dame.

1.4.14 La Liste Des Atomes Standard

Pour la connaitre chantez quelque chose comme :

```
(DE FORMAT (L NC NAL)
  (LET ((X 1) (NN NAL))
       (IF (NULL L) (OR (EQ NN NAL) (TERPRI))
           (TTAB X)
           (PRIN1 (NEXTL L)) (SETQ X (+ X NC))
           (DECR NN)
           (COND ((ZEROP NN)
                  (TERPRI) (SETQ NN NAL X 1)))
           (SELF X NN))))

; L   = une liste d'atomes
  NC  = espace inter-colonnes
  NAL = nombre d'atomes par ligne ;

 (LET ((RES NIL) (X (LOC NIL)))
      (IF (GT X (LOC 'SYS3)) (FORMAT (REVERSE RES) 10 7)
          (SELF (CONS (VAG X) RES) (+ X 6))))
```

1.4.15 La Bibliotheque Initiale

Un fichier standard nomme INIVLI-:L est automatiquement charge toutes les fois que vous rentrez dans VLISP 16 a froid, par >LISP sous TSF. Ce fichier est tres indispensable a la bonne marche du systeme. Il contient des definitions de fonctions, de macros, de macro-caracteres, toutes sortes de choses interessantes.

Examinez-le donc avec EDIT16, ainsi que les autres fichiers standard, tels que PRETTY-:L, TRACEF-:L, DEBUGG-:L ...

Ces fichiers sont partages par TOUS les utilisateurs de TSF. Si vous voulez les modifier ou les completer, faites donc une copie prealable et modifiez votre copie de la maniere qui vous plaira et tenez-moi au courant en decrivant brievement vos innovations dans le fichier d'informations VLIDOC-:L, prevu a cet effet.

CHAPITRE 2

FONCTIONS DE DEFINITION ET TYPES DE FONCTIONS

VLISP 16 comporte 7 types de fonctions :

SUBR NSUBR FSUBR EXPR NEXPR FEXPR MACRO

Les {SUBR, NSUBR, FSUBR} sont des fonctions standard, ecrites en langage-machine SOLAR 16.

Les {EXPR, NEXPR, FEXPR, MACRO} sont des fonctions definies par l'utilisateur en langage VLISP.

2.1 LAMBDA-FONCTIONS ET FONCTIONS DEFINIES

Une fonction en VLISP 16 pourra etre literale (ou anonyme, ou encore immediate) et prendra alors la forme d'une LAMBDA-EXPRESSION (Lambda-Fonction) :

```
(LAMBDA <lpf> . <corps>)

avec :  <lpf> = liste de parametres formels.
      <corps> = <e1> <e2> ... <eN>
                ou les <eI> sont des expressions qui
                seront evaluees en sequence, la valeur
                ramenee etant celle de <eN>.
```

Une fonction pourra etre egalement definie (ou nommee) et sera introduite sous la forme de :

```
(DE <nom> <lpf> . <corps>)
ou
(DF <nom> <lpf> . <corps>)
ou
(DM <nom> <lpf> . <corps>)
```

Une lambda-expression de la forme precedente sera automatiquement generee par le systeme et associee au <nom> sur sa P-liste.

Une liste de parametres formels aura la forme :

```
(<pf1> <pf2> ... <pfN>)
ou
(<pf1> <pf2> ... <pfN-1> . <pfN>)
ou
<pf>
```

Dans le second cas le parametre <pfN> sera lie avec le RESTE des valeurs des arguments de l'appel i.e. avec le N-ieme CDR de la liste des valeurs.

EX :

```
((LAMBDA (X Y . Z) [X Y Z]) 1 2 3 4)
                  -> (1 2 (3 4))

(DE FOO (X Y . Z)
   [X Y Z])
(FOO 1 2 3 4)  ->  (1 2 (3 4))
```

Dans le troisieme cas, <pf> sera liee avec la LISTE des valeurs des arguments a l'appel.

EXEMPLE :

```
((LAMBDA X [X X]) 1 (ADD1 1) (SUB1 4))
                    -> ((1 2 3) (1 2 3))

(DE FOO X (APPEND (CDR X) [(CAR X)]))

(FOO (ADD1 0) 2 (SUB1 4))
                    -> (2 3 1)
```

Si une fonction (standard ou definie par l'utilisateur) n'a pas assez d'arguments fournis a l'appel, les arguments manquants sont supposes etre NIL. La fonction ne peut pas distinguer entre NIL donne comme argument et pas d'argument du tout (ceci ne valant bien entendu que pour un facteur droit de la liste d'arguments).

EXEMPLE :

```
(FOO)          aura le meme effet que (FOO NIL)
(SETQ X NIL)   aura le meme effet que (SETQ X)
```

Si l'appel de la fonction comporte trop d'arguments (pour les EXPR et les SUBR), ils sont evalues mais ignores par la fonction. Donc, si on a une fonction <f> a M arguments, et qu'on evalue

```
(<f> <arg1> ... <argN>)  avec N > M
```

les arguments <argM+1>, <argM+2>, ... , <argN> seront evalues mais leurs valeurs seront ignorees.

2.2 TYPES DES FONCTIONS

2.2.1 LES EXPRs

A l'appel d'une fonction de type EXPR, tous les arguments sont evalues (de gauche a droite), puis les valeurs des arguments sont liees une a une (et de gauche a droite) avec les parametres formels de la fonction. S'il y a plus de valeurs que de parametres formels, les valeurs restantes sont ignorees. S'il y a plus de parametres que de valeurs, ceux-ci sont alors lies par defaut a la valeur NIL, et font ainsi office de variables locales.
A la suite de quoi le corps de la fonction est evalue.

EX :

```
(DE FOO (X Y) [X Y])
(FOO (ADD1 1) (SUB1 1))  ->  (2 0)
```

2.2.2 LES NEXPRs

A l'appel d'une fonction de type NEXPR, tous les arguments sont evalues (de gauche a droite), puis les valeurs des arguments sont rassemblees en une liste qui est liee avec l'unique parametre formel de la fonction.
A la suite de quoi le corps de la fonction est evalue.

EX :

```
(DE FOO X (REVERSE X))
(FOO (ADD1 1) (SUB1 1))  ->  (0 2)
```

2.2.3 LES FEXPRs

A l'appel d'une fonction de type FEXPR, aucun des arguments n'est evalue. Le CDR de l'appel est lie au premier parametre formel de la fonction. Les autres parametres formels (s'il y en a) seront lies a NIL et feront ainsi office de variables locales.
A la suite de quoi le corps de fonction est evalue.

EX :

```
(DF FOO (X Y Z) [X Y Z])
(FOO (ADD1 1) (SUB1 1))
        ->
(((ADD1 1) (SUB1 1)) NIL NIL)
```

2.2.4 LES MACROs

L'interprete VLISP 16 reconnait le type MACRO. Une macro est un atome qui possede sur sa P-liste une lambda-expression sous l'indicateur MACRO.
A l'evaluation d'un appel dont le CAR est une macro, EVAL lancera d'abord la fonction associee a cette macro, avec l'appel comme argument, puis re-evaluera la valeur retournee de cette premiere evaluation. L'evaluation d'une macro se fait donc en deux temps. Cette evaluation est nommee macro-generation.
C'est l'appel de la macro tout entier qui est passe en argument, il est donc possible de modifier physiquement cet appel a la premiere evaluation de la macro.
Ces MACROs ne doivent pas etre confondues avec les macro-caracteres d'entree (MCHAR) qui ne sont actifs que pendant les lectures.

EX :

De nouvelles structures de controle peuvent ainsi etre definies par macro-generation.
Voici la structure REPEATWHILE :

```
(REPEATWHILE <s1> ... <sN> <s-test>)
```

qui evalue en sequence <s1> ... <sN> tant que la valeur de la derniere expression <s-test> est differente de NIL avec le test en fin de sequence.

```
(DM REPEATWHILE (CALL) (RPLACB CALL
   ['WHILE (CONS 'PROGN (CDR CALL))]))
```

2.3 FONCTIONS DE DEFINITION

La definition de fonction est effectuee par modification physique de P-liste d'atome.

(DE <a> <la> <s1> ... <sN>) [FSUBR]

definit une fonction de type EXPR avec :
<a> (atome literal), comme nom de la fonction.
<la> (atome literal ou liste), comme argument(s) de la fonction.
<s1> ... <sN>, comme corps de la fonction.
DE ramene <a> en valeur et est equivalent a :
(PUT '<a> (LAMBDA <la> <s1> ... <sN>) EXPR).

(DF <a> <la> <s1> ... <sN>) [FSUBR]

equivalent a DE, mais la fonction ainsi definie est du type FEXPR.
DF ramene <a> en valeur et est equivalent a :
(PUT '<a> (LAMBDA <la> <s1> ... <sN>) FEXPR).

(DM <a> <la> <s1> ... <sN>) [FSUBR]

equivalent a DF et DE, mais la fonction ainsi definie est du type MACRO.
DM ramene <a> en valeur et est equivalent a :
(PUT '<a> (LAMBDA <la> <s1> ... <sN>) MACRO).

Il va de soi qu'on peut re-definir une fonction deja definie (de meme qu'une fonction standard).

EXEMPLE :

```
?(DE FOO (X) (BAR (REVERSE X)))
 FOO
?(DE FOO (X Y) (BAR (REVERSE Y) (REVERSE X)))
 FOO
```

Notez que si une fonction d'un type (EXPR, FEXPR, MACRO) est redefinie sous un autre type, c'est la premiere definition qui demeurera connue de l'interprete. La ou les definitions suivantes ne sont cependant pas perdues et sont toujours sur la P-liste de l'atome-nom de la fonction.

CHAPITRE 3

LES FONCTIONS STANDARD

3.1 LES FONCTIONS INTERPRETE

(EVAL <s>) [SUBR a 1 argument]

C'est la fonction principale de l'interprete. EVAL ramene la valeur de l'evaluation de l'argument <s>.

ex : (EVAL '(ADD1 55)) ☞ 56

(APPLY <fn> <l>) [SUBR a 2 arguments]

ramene la valeur de l'application de la fonction <fn> a la liste d'arguments <l>.

ex : (APPLY 'PLUS [5 (ADD1 8) (REM 10 3)]) ☞ 15

(EVLIS <l>) [SUBR a 1 argument]

ramene une liste composee des valeurs des evaluations de tous les elements de la liste <l>.

ex : (SETQ L '((ADD1 5) (ADD1 7) (ADD1 9)))
☞ ((ADD1 5) (ADD1 7) (ADD1 9))
(EVLIS L) ☞ (6 8 10)

(EPROGN <l>) [SUBR a 1 argument]

evalue tous les elements de la liste <l>. EPROGN ramene en valeur la valeur de la derniere evaluation (i.e. celle du dernier element de <l>).

ex : (SETQ L '((PRIN1 1)(PRIN1 2)(PRIN1 3)))
☞ ((PRIN1 1)(PRIN1 2)(PRIN1 3))
(EPROGN L) 1 2 3 ☞ 3

(PROGN <s1> ... <sN>) [FSUBR]

evalue les differentes expressions <s1> ... <sN>. PROGN ramene la valeur de la derniere evaluation (i.e. celle de <sN>).

```
ex : (PROGN (PRIN1 1)(PRIN1 2)(PRIN1 3))  1 2 3  ☞  3
```

(QUOTE <s>) [FSUBR]

ramene la S-expression <s> non-evaluee. Cette fonction est utilisee comme argument de fonctions de type SUBR dont on ne desire pas evaluer les arguments. Il existe un macro-caractere standard qui facilite cette ecriture, la caractere quote (l'apostrophe) ' .

```
ex : (QUOTE (ADD1 4))    ☞  (ADD1 4)
     '(A (B C))          ☞  (A (B C))
     'A                  ☞  A
     ''A                 ☞  (QUOTE A)
```

(LAMBDA <s> <s1> ... <sN>) [FSUBR]

la valeur d'une forme LAMBDA est cette forme elle-meme. Cette nouvelle fonction a ete rajoutee pour eviter de "quoter" les lambda-expressions explicites des fonctionnelles.

```
ex : (LAMBDA (X) X)     ☞   (LAMBDA (X) X)
     (MAPC '(A B C) (LAMBDA (X) (PRIN1 X))) A B C  ☞  NIL
     ((LAMBDA (X) (X X)) (LAMBDA (X) (X X)))
                          ->
                         boucle
                         mais

     ((LAMBDA (X) [X [QUOTE X]]) (LAMBDA (X) [X [QUOTE X]]))
                       ->
     ((LAMBDA (X) [X [QUOTE X]]) (LAMBDA (X) [X [QUOTE X]]))
                         ne boucle pas
```

Une interessante application de cette possiblilite est la fameuse fonction :

```
(DE LIT (L E F)
  (IF L (F (NEXTL L) (LIT L E F))
      E))
```

LIT permet d'ecrire des definitions tres elegantes et concises.

```
(APPEND X Y)  =  (LIT X Y 'CONS)
(MAPCAR L F)  =  (LIT L NIL
                      (LAMBDA (X Y) (CONS (F X) Y)))
```

```
(POWERSET '(A B C))
   ☞ ((A B C) (A B) (A C) (A) (B C) (B) (C) ())
avec
   (DE POWERSET (L)
     (LIT L [NIL]
          (LAMBDA (A B)
            (LIT B B
                 (LAMBDA (C D) (CONS (CONS A C) D))))))
aussi
   (PROCAR '((A B) (C D E) (F G))
   ☞ ((A C F) (A C G) (A D F) ... (B E G))
avec
   (DE PROCAR (L)
     (LIT L [NIL]
          (LAMBDA (A B)
            (LIT A NIL
                 (LAMBDA (C D)
                   (LIT B D
                        (LAMBDA (E F)
                          (CONS (CONS C E) F))))))))
```

(SELF <s1> ... <sN>) [SUBR a N arguments]

appelle la derniere lambda-expression dynamiquement active avec <s1> ... <sN> pour arguments. Cette fonction permet de resoudre le probleme des anciennes fonctions LABEL sans avoir a nommer obligatoirement les lambda-expressions explicites recursives.

```
ex : ((LAMBDA (L)
       (IF (NULL (CDR L))
           (CAR L)
           (SELF (CDR L))))
     '(A B C D))
    ☞ D

    (DE TRI-INSERTION (L)
      (IF L
        (LET ((X (NEXTL L)) (L (SELF L)))
             (IF (OR (NULL L) (LE X (CAR L)))
                 (CONS X L)
                 (CONS (NEXTL L) (SELF X L))))))

    (TRI-INSERTION '(5 3 4 2 3 1)) ☞ (1 2 3 3 4 5)
```

```
(LET ((<at1> <s1>) ...   (<atN> <sN>)) .  <l>) [MACRO]
```

evalue les <s1> ... <sn> dans l'environnement de l'appelant, puis lie <s1> a la variable <at1> etc..., et dans ce nouvel environnement evalue sequentiellement la liste d'expressions <l>.

LET est equivalente a

```
((LAMBDA (<at1> ... <atN>) . <l>) <s1> ... <sN>)
```

Le premier des exemples precedents illustrant le SELF s'ecrira ici

```
(LET ((L '(A B C D)))
       (IF (CDR L) (SELF (CDR L)) (CAR L)))
```

Pour calculer les nombres de Fibonacci en temps lineaire :

```
(DE FIB-LIN (N)
  (IF (< N 2) 1
      (LET ((N N) (L '(1 1))) (COND
        ((CDDR L) (IF (= N 2) (CADDR L)
                      (SELF (1- N) (CDR L))))
        ((= N 2) (CADR (RPLACD (CDR L)
                              [(+ (CAR L) (CADR L))])))
        (T (SELF (1- N) (RPLACD (CDR L)
                              [(+ (CAR L) (CADR L))])))))))
```

Faites donc un Pretty-Print de FIB-LIN, apres l'appel (FIB-LIN 10).

L'idee de cette superbe fonction est due a John Mc Carthy.

3.2 LES PREDICATS DE BASE

3.2.1 Tests Sur Les Types

(ATOM <s>) [SUBR a 1 argument]

teste si <s> est un atome, i.e. un atome literal, ou un nombre. ATOM ramene T si ce test est verifie, NIL dans le cas contraire.

```
ex : (ATOM 'ARGH)     ☞ T
     (ATOM 42)        ☞ T
     (ATOM '(A B))    ☞ NIL
     (ATOM NIL)       ☞ T
```

(LISTP <s>) [SUBR a 1 argument]

teste si <s> est une liste. LISTP ramene T si le test est verifie et NIL dans le cas contraire.

```
ex : (LISTP 'ARGH)    ☞ NIL
     (LISTP 44)       ☞ NIL
     (LISTP '(A B))   ☞ T
     (LISTP NIL)      ☞ NIL
```

(NULL <s>) [SUBR a 1 argument]

teste si <s> est egal a NIL. NULL ramene T si le test est verifie et NIL dans le cas contraire.

```
ex : (NULL NIL)   ☞ T
     (NULL T)     ☞ NIL
     (NULL)       ☞ T
```

(NOT <s>) [SUBR a 1 argument]

cette fonction est identique a NULL.

3.2.2 Les Comparaisons

(EQ <s1> <s2>) [SUBR a 2 arguments]

sert a tester 2 atomes (de n'importe quel type : atome literal, nombre). EQ ramene T si les 2 atomes <s1> et <s2> sont egaux et NIL s'ils ne le sont pas.

Rappellons que :
- 2 atomes literaux sont egaux s'ils ont le meme nom externe
- 2 nombres sont egaux s'ils ont la meme valeur.

Si <s1> et <s2> sont des listes, EQ teste si <s1> et <s2> ont la meme adresse physique (l'auteur est totalement conscient du NON SEQUITUR de cette specification).

```
ex : (EQ 'A (CAR '(A)))        ☞  T
     (EQ (ADD1 119) 120)       ☞  T
     (EQ '(A B) '(A B))        ☞  NIL
     (EQ)                      ☞  T
```

(NEQ <s1> <s2>) [SUBR a 2 arguments]

est equivalent a (NOT (EQ <s1> <s2>)).

(EQUAL <s1> <s2>) [SUBR a 2 arguments]

est la fonction de comparaison la plus generale. EQUAL teste si <s1> et <s2> ont la meme structure. Si le test est verifie, EQUAL ramene T et dans le cas contraire EQUAL ramene NIL.

```
ex : (EQUAL '(A (B . C) D) '(A (B . C) D))  ☞  T
```

Un tres vieux generateur de permutations en ordre lexicographique :

```
(P '(A B C))
☞ ((A B C) (A C B) (B C A) (B A C) (C A B) (C B A))
```

avec

```
(DE P (X)
  (IF (NULL X) [[]]
      (F X)))

(DE F (L)
  (APPEND (MAPCAR (P (CDR L))
                  (LAMBDA (X) (CONS (CAR L) X)))
          (LET ((Y (APPEND (CDR L) [(CAR L)])))
            (IF (EQUAL X Y) NIL
                (F Y)))))
```

ATTENTION :
Dans le cas de 2 listes, EQ teste si celles-ci ont la meme ADRESSE d'implantation en memoire (i.e. s'il s'agit bien du MEME objet). En revanche EQUAL teste si les 2 listes sont structuralement identiques, meme a etre implantees a des adresses distinctes.

(SORT <a1> <a2>) [SUBR a 2 arguments]

ramene T si le P-name de <a1> est plus petit ou egal (lexicographiquement) au P-name de <a2>, sinon ramene NIL. Cette foncion est utilisee pour realiser des tris alphabetiques.

```
ex : (SORT 'A 'A)         ☞ T
     (SORT 'B 'A)         ☞ NIL
     (SORT 'A 'B)         ☞ T
     (SORT 'ZZZ 'ZZZZ)    ☞ T
```

3.3 LES FONCTIONS DE CONTROLE

3.3.1 Les Fonctions De Controle De Base

(OR <s1> ... <sN>) [FSUBR]

evalue successivement les differentes expressions <s1> ... <sN> jusqu'a ce que l'une de ces evaluations ait une valeur differente de NIL. OR ramene cette valeur. OR permet de construire une structure de controle de type :

si non <s1> alors si non <s2> alors ... sinon <sN> fsi

```
ex : (OR NIL NIL 2 3)   ☞ 2
     (OR)               ☞ NIL
```

(AND <s1> ... <sN>) [FSUBR]

evalue successivement les differentes expressions <s1> ... <sN>. Si la valeur d'une de ces evaluations est egale a NIL, AND ramene NIL sinon AND ramene la valeur de la derniere evaluation <sN>. AND permet de construire une structure de controle de type :

si <s1> alors si <s2> alors ... sinon <sN> fsi

```
ex : (AND 1 2 3 4)    ☞ 4
     (AND 1 2 () 4)   ☞ NIL
     (AND)            ☞ NIL
```

AND et OR sont iteratifs au sens de VLISP

ex : Pour savoir si l'atome X a une occurence dans L

```
(DE SEARCH (X L)
  (OR (AND (ATOM L) (EQ X L))
      (AND (LISTP L) (SEARCH X (NEXTL L)))
      (AND (LISTP L) (SEARCH X L))))
```

L'appel (SEARCH X L)

est iteratif.

(IF <s1> <s2> <s3> ... <sN>) [FSUBR]

si la valeur de l'evaluation de <s1> est differente de NIL, IF ramene la valeur de l'evaluation de l'expression <s2>, sinon IF evalue en sequence les expressions <s3> ... <sN> et ramene la valeur de la derniere evaluation <sN>. IF permet de construire une structure de controle de type :

si <s1> alors <s2> sinon <s3>; ... ;<sN> fsi

ex :
```
(IF T 1 2 3)      ☞ 1
(IF NIL 1 2 3)    ☞ 3

(DE ACK (X Y)
  (IF (= X 0) (1+ Y)
      (ACK (1- X)
           (IF (= Y 0) 1
               (ACK X (1- Y))))))
```

(COND <l1> ... <lN>) [FSUBR]

est l'instruction conditionnelle la plus generale. Les differents arguments <l1> ... <lN> sont des listes appellees clauses qui ont la structure suivante :

(<ss> <s1> ... <sN>)

COND va choisir une seule de ces clauses : celle dont l'evaluation de son premier element <ss> est differente de NIL. COND evalue alors les differentes expressions <s1> ... <sN> et ramene la valeur de la derniere evaluation <sN>. Si la clause choisie n'a qu'un element <ss>, COND ramene la valeur de l'evaluation de <ss> (i.e. la valeur qui a declenche la selection de cette clause). COND permet de construire des structures de controle de type :

si ... alors ... sinon si ... alors ... ... fsi

Si aucune clause n'est choisie, COND ramene NIL.

Ainsi

```
(COND
  (p1 e11 e12 e13)
  (p2 e21 e22)
  (p3)
  (p4 e41))
```

sera considere comme equivalent a :

```
(COND
  (p1 (PROGN e11 e12 e13))
  (p2 (PROGN e21 e22))
  ((SETQ aux p3) aux)
  (p4 e41)
  (T NIL))
```

EXEMPLES :

```
(COND (NIL 1 2) (T 3 4 5))   ☞ 5

(COND ((LT 5 4) 'FOO) ((NULL 'A) 'BAR))  ☞ NIL

(COND ((ZEROP X) 'ZERO)
      ((ODDP X)  'IMPAIR)
      ((EVENP X) 'PAIR)
      (T 'What?!?))
```

(SELECTQ <s> <l1> ... <lN> <lf>) [FSUBR]

comme pour la fonction COND, SELECTQ va choisir une des clauses <l1> ... <lN>. Le selecteur de ces clauses est la valeur de l'evaluation de <s>, la selection s'effectue par comparaison du selecteur avec le CAR (non evalue) de la clause qui doit etre un atome (en utilisant le predicat EQ).
Des qu'une clause est choisie, SELECTQ evalue le reste de la clause et ramene la valeur de la derniere evaluation.
Si aucune des clauses <l1> ... <lN> n'est choisie, SELECTQ evalue automatiquement la derniere clause <lf> et ramene la valeur de (PROGN . <lf>). SELECTQ permet donc de construire des aiguillages sur valeurs constantes.

ex :
```
(SELECTQ 'ROUGE
   (VERT 'ESPOIR)
   (ROUGE 'OK)
   ('NON))
☞ OK
(SELECTQ 'BLANC
   (VERT 'ESPOIR)
   (BLEU 'OK)
   ('NON))
☞ NON
```

(WHILE <s> <s1> ... <sN>) [FSUBR]

tant que la valeur de l'evaluation du test <s> est differente de NIL, WHILE va evaluer en sequence les differentes expressions <s1> ... <sN>. WHILE ramene toujours NIL en valeur (qui est la derniere evaluation de <s> qui fait sortir de la boucle WHILE). Cette fonction permet de construire des boucles conditionnelles d'une maniere fort commode, ainsi que des boucles infinies en utilisant la forme : (WHILE T ...).

```
ex : (SETQ L '(A B C D))                     ☞ (A B C D)
     (WHILE L (PRIN1 (NEXTL L))) A B C D  ☞ NIL
```

La forme (REPEATUNTIL <s1> ... <sN> <s>) avec le test <s> a la fin se definira comme :

```
(DM REPEATUNTIL (CALL) (RPLACB CALL
  ['WHILE (CONS 'NOT (CONS 'PROGN (CDR CALL)))]))
```

(WHILE <test> <s1> ... <sN>)

est equivalent a

(LET () (COND (<test> <s1> ... <sN> (SELF))))

3.3.2 Les Fonctions D'echappement

(LESCAPE <s1> ... <sN>) [FSUBR]

evalue les differentes expressions <s1> ... <sN> en sequence et ramene la valeur de la derniere evaluation <sN>. De plus LESCAPE fait sortir de la derniere lambda-expression active (fonction utilisateur ou lambda-expression explicite) avec pour valeur la valeur de <sN>.

```
ex : (LET ((X 3) (Y NIL))
     (WHILE T
        (IF (ZEROP X) (LESCAPE Y)
            (SETQ Y (CONS X Y) X (SUB1 X)))))
     ☞ (1 2 3)
```

(ESCAPE <at> <s1> ... <sN>) [FSUBR]

est la fonction de controle la plus puissante compatible avec la structure recursive de VLISP. <at> est un atome literal qui devient le nom d'une fonction d'echappement, puis les differentes expressions <s1> ... <sN> sont evaluees en sequence. Si au cours de ces evaluations une forme de type (<at> <ss1> ... <ssN>) est rencontree, les differentes expressions <ss1> ... <ssN> sont evaluees et ESCAPE ramene la valeur de la derniere evaluation (i.e. celle de <ssN>). Si une telle forme n'est pas rencontree, ESCAPE ramene la valeur de l'evaluation de <sN>.

```
ex : (ESCAPE EXIT
       (MAPC '(A B 2 C)
         (LAMBDA (X) (AND (NUMBP X) (EXIT 'OUI))))
       'NON)
     ☞ OUI
```

Pour ceux qui aiment, il est tres facile d'exprimer avec ESCAPE les constructions

(CATCH <x> <une-etiquette>)

et

(THROW <y> <la-meme-etiquette>)

par

```
(DM CATCH (CALL) (RPLACB CALL
   ['ESCAPE (CADDR CALL) (CADR CALL)]))

(DM THROW (CALL) (RPLACB CALL
   [(CADDR CALL) (CADR CALL)]))
```

Impossible en revanche d'exprimer ESCAPE en termes de CATCH et THROW.

3.3.3 Les Fonctions De Controle De Type PROG

Ce type de structure de controle permet a l'usager conservateur d'ecrire des sequences VLISP sans structure, vertu nocturne partagee avec certains autres langages.
On peut se "brancher a des etiquettes" (fonctions GO et GOTO) et sortir d'un corps de PROG comme en FORTRAN (avec la fonction eternelle : RETURN).
Ce type de structure de controle a ete conserve dans un souci de compatibilite avec certains autres systemes LISP qui ne possedent que ce type de structure de controle, les fonctions d'echappement des systemes VLISP etant a la fois plus puissantes et plus rapides a l'interpretation.

(PROG <l> <s1> ... <sN>) [FSUBR]

<l> est une liste d'atomes literaux qui servent de variables locales dans la portee du PROG. Elles sont liees a la valeur NIL a l'entree du PROG et restaurees a leurs anciennes valeurs au retour du PROG. Cette protection des variables locales ne porte que sur leurs C-valeurs. Cette liste de variables locales peut etre vide mais ne peut pas etre omise. <s1> ... <sN> est une liste d'expressions qui sont evaluees en sequence. Si dans cette liste se trouvent des atomes, ils sont consideres comme des etiquettes et ne sont donc pas evalues. La valeur d'un PROG, s'il n'est pas interrompu par un RETURN, est la valeur de l'evaluation de <sN> (qui ne doit PAS etre une etiquette).

(GO <a>) [FSUBR]

<a> est un atome literal et doit etre le nom d'une etiquette existante du dernier PROG actif. L'evaluation de la forme (GO <a>) fait reprendre l'evaluation a la forme qui suit l'etiquette <a>.

(GOTO <s>) [SUBR a 1 argument]

est identique a la fonction GO mais l'argument <s> est evalue et doit ramener un atome en valeur sous peine de declencher l'erreur A6.

(RETURN <s>) [SUBR a 1 argument]

sort du PROG actif. La valeur du PROG est celle de l'evaluation de <s>.

3.3.4 Les Fonctionnelles

Toutes ces fonctions sont de type SUBR et utilisent des fonctions <fn> en argument. Ces fonctions doivent etre de type SUBR ou EXPR mais seuls leurs premiers arguments seront lies a des objets variables, tous les autres seront lies a NIL.

(MAPxxx <l> <fn>) [SUBR a 2 arguments]

Elles permettent d'appliquer la fonction <fn> sur certaines composantes de la liste <l>.

applique la fonction <fn> sur :			ramene en valeur
<l> puis sur ses CDR successifs	les CAR successifs de <l>	<l> puis sur toutes ses sous-structures	
MAP	MAPC		NIL
	MAPCAR		la liste des valeurs de toutes les applications
			la liste des valeurs differentes de NIL de toutes les applications

```
ex : (MAP  '(A (B C) D) 'PRINT)
     (A (B C) D)
     ((B C) D)
     (D)
     ☞ NIL

     (MAPC '(A (B C) D) 'PRINT)
     A
     (B C)
     D
     ☞ NIL

     (MAPCAR [1 2 3 4] (LAMBDA (X) (+ X 5)))
     ☞ (6 7 8 9)
```

3.4 LES FONCTIONS DE RECHERCHE

3.4.1 Les Recherches Sur Des Objets LISP

(CAR <s>) [SUBR a 1 argument]

si <s> est un atome literal, ramene sa C-valeur.
si <s> est une liste, ramene son premier element.
Le CAR d'un nombre est indetermine.

```
ex : (CAR '(A B C))       ☞  A
     (SETQ VA '(U P))     ☞  (U P)
     (CAR 'VA)            ☞  (U P)
     (CAR VA)             ☞   U
```

(CDR <s>) [SUBR a 1 argument]

si <s> est un atome literal, ramene sa P-liste.
si <s> est une liste, ramene cette liste sans son premier element.
Le CDR d'un nombre est indetermine.

```
ex : (CDR '(A B C))         ☞  (B C)
     (PUT 'VA '(U P) 'I)    ☞  VA
     (CDR 'VA)              ☞  (I (U P))
```

(C...R <s>) [SUBR a 1 argument]

les 14 combinaisons de CAR et de CDR imbriques sont disponibles.
(CADR <s>) est equivalent a (CAR (CDR <s>))
(CDAAR <s>) est equivalent a (CDR (CAR (CAR <s>)))
(CADDR <s>) est equivalent a (CAR (CDR (CDR <s>)))

```
ex:   (DE FOO (L)
        (IF (CDR L) (LET ((L (CADR L)) (R [(CAR L)]))
                         (IF (CDR L)
                             (SELF (CADR L) [R (CAR L)])
                             (CONS R L)))
           L))

      (FOO '(A))           ☞  (A)
      (FOO '(A (B)))       ☞  ((A) B)
      (FOO '(A (B (C))))   ☞  (((A) B) C)
```

(MEMQ <a> <l>) [SUBR a 2 arguments]

si l'atome <a> est un element de la liste <l>, ramene la partie de la liste <l> commencant a l'atome <a>, sinon ramene NIL. Cette fonction utilise le predicat EQ pour tester la presence de l'atome <a> dans la liste.

Si <a> est une liste, MEMQ teste si <a> est un CAR de <l>.

```
ex : (MEMQ 'C '(A B C (X Y))    ☞  (C (X Y))
     (MEMQ 'Y '(A B C (X Y))    ☞  NIL
```

Nous laisserons au lecteur le soin d'etablir l'intention de la fonction suivante :

```
(DE SKE (L)
   (LET ((L L) (R NIL)) (COND
      ((ATOM L) NIL)
      ((MEMQ L R) T)
      (T (OR (SELF (CAR L) (CONS L R))
             (SELF (CDR L) (CONS L R))))))))
```

(MEMBER <s> <l>) [SUBR a 2 arguments]

si l'expression quelconque <s> est un element de la liste <l>, ramene la partie de <l> commencant a l'expression <s>, sinon ramene NIL. Cette fonction est identique a la fonction MEMQ mais utilise le predicat EQUAL.

```
ex : (MEMBER '(B C) '((A (B C)) (B C) D))  ☞  ((B C) D)
```

(LENGTH <l>) [SUBR a 1 argument]

ramene le nombre d'elements de la liste <l>.

```
ex :  (LENGTH '(A (B C) D))   ☞  3
      (LENGTH NIL)            ☞  0
```

(NTH <n> <l>) [SUBR a 2 arguments]

ramene la partie de la liste <l> commencant a son <n>ieme element (i.e. son <n>-1 eme CDR). Si (LENGTH <l>) est inferieure a <n>, NTH ramene NIL. Si <n> <= 1, NTH ramene <l> en entier.

```
ex : (NTH 3 '(A B C D E))  ☞  (C D E)
```

3.5 FONCTIONS DE CREATION DE LISTES ET D'ATOMES

(CONS <s1> <s2>) [SUBR a 2 arguments]

construit une liste dont le premier element est <s1> et le reste la liste <s2>. Si <s2> est un atome, CONS produit la paire pointee (<s1> . <s2>).

```
ex : (CONS 'A '(B C))        ☞  (A B C)
     (CONS 1 2)              ☞  (1 . 2)
     (CONS 'A)               ☞  (A)
     (CONS)                  ☞  (NIL)
```

On notera que CONS peut etre defini avec LIST et APPEND comme

```
(CONS x y)  =  (APPEND [x] y)
```

(MCONS <s1> <s2> ... <sN>) [SUBR a N arguments]

permet le CONS multiple.
L'appel

```
(MCONS <s1> <s2> ... <sN-1> <sN>)
```

correspond a

```
(CONS <s1> (CONS <s2> ... (CONS <sN-1> <sN>) ... ))
```

```
ex : (MCONS 'A 'B 'C)          ☞  (A B . C)
```

Si MCONS etait une macro elle serait definie comme :

```
(DM MCONS (CALL)
  (AND (CDDDR CALL)
       (RPLACD CALL
               [(CADR CALL)
                (CONS 'MCONS (CDDR CALL))]))
  (RPLACA CALL 'CONS))
```

(LIST <s1> ... <sN>) [SUBR a N arguments]

ramene la liste des valeurs des differentes expressions :

```
<s1> ... <sN>
```

En termes de CONS, l'appel

```
(LIST <s1> <s2> ... <sN>)
```

est equivalent a

```
(CONS <s1> (CONS <s2> ... (CONS <sN> NIL) ...))
```

Il existe une ecriture abregee de la fonction LIST qui utilise les macros-caracteres dits crochets carres :

```
[<s1> ... <sN>]
```

C'est l'ecriture generalement adoptee dans ce manuel.

```
ex : (LIST 'A 'B 'C)      ☞  (A B C)
     '[45 (ADD1 X) L]     ☞  (LIST 45 (ADD1 X) L)
     (LIST)               ☞  NIL
     [[]]                 ☞  (NIL)
```

(SUBST <s1> <at> <l>) [SUBR a 3 arguments]

fabrique une nouvelle copie de toute la liste <l> en substituant a l'atome <at> l'expression <s1> a chacune de ses occurences.

```
ex : (SUBST '(A B) 'C '(C (D E) (C (A B))))
     ☞  ((A B) (D E) ((A B) (A B)))
```

Pour effectuer une recopie de TOUS les niveaux de la liste L faire :

```
(SUBST () () L)
```

Pour effectuer une copie du seul premier niveau de la liste L faire :

```
(APPEND L)
```

Pour n'effectuer que les copies necessaires, redefinissez SUBST ainsi :

```
(DE SUBST (X Y E)
  (IF (ATOM E) (IF (EQ E Y) X E)
      (LET ((E1 (SUBST X Y (CAR E)))
            (E2 (SUBST X Y (CDR E))))
        (IF (AND (EQ E1 (CAR E)) (EQ E2 (CDR E)))
            E
            (CONS E1 E2)))))
```

Enfin, pour faire une copie d'une liste cyclique ou partagee :

```
(DE FCOPY (L ;; D)
  (LET ((L L) (NEW)) (COND
    ((ATOM L) L)
    ((CASSQ L D))
    (T (SETQ NEW [NIL] D (CONS (CONS L NEW) D))
       (RPLACD (RPLACA NEW (SELF (NEXTL L)))
               (SELF L))))))
```

(OBLIST) [SUBR a 0 argument]

ramene la liste de tous les atomes literaux crees au cours d'une session.

(GENSYM) [SUBR a 0 argument] ramene a chaque appel un NOUVEL atome de la forme : Gnnnn, dans laquelle les <n> sont des chiffres decimaux.

```
ex :    (GENSYM)  ☞  G0001     (au premier appel)
        (GENSYM)  ☞  G0002     (a l'appel suivant)
        etc ...
```

(STATUS 6 <code-ascii>) [SUBR a 2 arguments]

permet de creer des atomes mono-caractere ni imprimables ni introductibles sous EDIT16.

```
ex :    (STATUS 6 000CHH)  ☞   control-L
      ? (SETQ BELL (STATUS 6 0087HH))
        >beep<
      ? BELL
        >beep<
```

(REVERSE <s1> <s2>) [SUBR a 2 arguments]

ramene une copie inversee du premier niveau de la liste <s1>. Si le deuxieme argument est fourni, cette copie est NCONCee au second argument : l'appel correspond donc a (NCONC (REVERSE <s1>) <s2>).

```
ex : (REVERSE '(A (B C) D))             ☞  (D (B C) A)
     (REVERSE '((X Y) (U P)) '(G O))    ☞  ((U P) (X Y) G O)
```

Pour inverser physiquement une liste :

```
(DE FREVERSE (L ;; R)
  (IF L (FREVERSE (CDR L) (RPLACD L R))
      R))
```

REVERSE en VLISP 16 n'est PAS implemente ainsi :

```
(DE REV (L)
  (IF (NULL (CDR L)) L
      (CONS (CAR (REV (CDR L)))
            (REV (CONS (CAR L)
                  (REV (CDR (REV (CDR L)))))))))
```

(APPEND <l> <s>) [SUBR a 2 arguments]

ramene la concatenation d'une copie du premier niveau de la liste <l> a l'expression <s>. Si <s> n'est pas fourni, (APPEND <l>) ramene simplement une copie du premier niveau de <l>.

```
ex : (APPEND '(A B C))            ☞  (A B C)
     (APPEND '(A B C) '(X Y))     ☞  (A B C X Y)
     (APPEND '(A B C) 'D)         ☞  (A B C . D)
```

APPEND est naturellement associative :

(APPEND x (APPEND y z)) = (APPEND (APPEND x y) z)

Cette propriete est aisement demontrable grace au programme CAN de Daniel GOOSSENS.

On notera egalement que :

```
(REVERSE (APPEND x y)) = (APPEND (REVERSE y) (REVERSE x))
 ainsi que
(REVERSE (REVERSE x))  = x
 ainsi que
(APPEND x y)           = (REVERSE (REVERSE x NIL) y)
```

C'est sur cette derniere propriete qu'est fondee la definition iterative d' APPEND :

```
(DE APPEND (X Y)
  (LET ((X X) (Y Y) (R NIL)) (COND
    (X (SELF (CDR X) Y (CONS (CAR X) R)))
    (R (SELF X (CONS (CAR R) Y) (CDR R)))
    (T Y))))
```

(DELETE <s> <l>) [SUBR a 2 arguments]

ramene une copie du premier niveau de la liste <l> dans laquelle toutes les occurences de l'expression <s> ont ete enlevees. Cette fonction utilise le predicat EQUAL.

ex : (DELETE '(X Y) '(Z (X Y) U (X Y) P)) ☞ (Z U P)

3.6 LES FONCTIONS DE MODIFICATIONS

Toutes les fonctions qui vont etre decrites doivent etre utilisees conformement au mode d'emploi, pour eviter de placer le systeme dans un etat de confusion dramatique car elles modifient physiquement les structures VLISP.

D'une maniere generale il est tres vivement souhaite de ne pas modifier :
- les constantes symboliques de l'interprete (NIL T LAMBDA ...) et d'une maniere generale tous les atomes de l'interprete.
- les nombres.

Si cette recommendation n'est pas suivie, les resultats seront terrifiants.

Pour ces fonctions l'argument <obj> represente soit un atome literal soit une liste. Modifier le CAR d'un atome revient a changer sa C-valeur, modifier son CDR revient a modifier sa P-liste.

(RPLACA <obj> <s>) [SUBR a 2 arguments]

remplace le CAR de <obj> par <s>. Ramene le nouvel <obj> en valeur.

```
ex : (SETQ X '(A B))               ☞  (A B)
     (RPLACA  X '(C))              ☞  ((C) B)
     X                             ☞  ((C) B)
     (RPLACA '(A B C) '(X Y))      ☞  ((X Y) B C)
```

(RPLACD <obj> <s>) [SUBR a 2 arguments]

remplace le CDR de <obj> par <s>. Ramene le nouvel <obj> en valeur.

```
ex : (RPLACD  X 'E)                ☞  ((C) . E)
     X                             ☞  ((C) . E)
     (RPLACD '(A B C) '(X Y Z))    ☞  (A X Y Z)
```

La fonction C-REVERSE inverse la liste circulaire L

```
(DE C-REVERSE (L)
  (AND L
    (LET ((C (CDR L)) (P L))
      (IF (NEQ L C) (SELF (CDR C) (RPLACD C P))
          (RPLACD L P)
          P))))
```

(RPLACB <obj> <l>) [SUBR a 2 arguments]

remplace le CAR de <obj> par le CAR ce <l> et le CDR de <obj> par le CDR de <l>.
En VLISP 16, RPLACB peut etre definie comme :

```
(DE RPLACB (OBJ L)
    (RPLACA (RPLACD OBJ (CDR L)) (CAR L)))
```

```
ex : (SETQ L1 '(A B C))       ☞  (A B C)
     (SETQ L2 L1)             ☞  (A B C)
     (RPLACB L1 '(X Y))       ☞  (X Y)
     L2                       ☞  (X Y)
```

RPLACB est une fonction privilegiee pour la definition de MACROS

ex : Consideree comme MACRO, LET est ainsi definie

```
(DM LET (CALL) (RPLACB CALL
  (CONS (MCONS LAMBDA (MAPCAR (CADR CALL) 'CAR)
                (CDDR CALL))
        (MAPCAR (CADR CALL) 'CADR))))
```

De meme, INCR peut etre definie

```
(DM INCR (CALL) (RPLACB CALL
  ['SETQ (CADR CALL) ['ADD1 (CADR CALL)]]))
```

(SET <obj> <s>) [SUBR a 2 arguments]

remplace le CAR de <obj> par <s> . Ramene en valeur <s>. A la valeur ramenee pres, (SET <obj> <s>) est equivalent a (RPLACA <obj> <s>).

```
ex : (SETQ L '(A B C))        ☞ (A B C)
     L                        ☞ (A B C)
     (SET L '(X Y))           ☞ (X Y)
     L                        ☞ ((X Y) B C)
     (SETQ A 1 X 'A)          ☞ A
     A                        ☞ 1
     (SET X 2)                ☞ 2
     A                        ☞ 2
     (SET L L)                ☞ ((((((((((((( ...
```

SET peut etre definie en terme de SETQ

```
(DE SET (X Y) (EVAL ['SETQ X [QUOTE Y]]))
```

(SETQ <at1> <s1> ... <atN> <sN>) [FSUBR]

<at1> ... <atN> sont des atomes literaux qui ne sont pas evalues; <s1> ... <sN> sont des expressions quelconques qui seront evaluees. SETQ est la fonction d'affectation la plus utilisee : chaque atome <at> est affecte par l'expression correspondante <s>. SETQ ramene <sN> en valeur.

```
ex : (SETQ L1 '(A B C))     ☞  (A B C)
     (SETQ L2 L1)           ☞  (A B C)
     (SETQ X 1 Y (1+ X))    ☞  2
```

SETQ peut etre definie en terme de SET

```
(DF SETQ (X)
  (LET ((VAR (CAR X)) (VAL (EVAL (CADR X)))
                (REST (CDDR X)))
    (SET VAR VAL)
    (IF REST (EVAL (CONS 'SETQ REST)) VAL)))
```

(NEXTL <at>) [FSUBR]

<at> (qui n'est pas evalue) doit etre un atome literal dont la valeur doit etre une liste. NEXTL ramene le CAR de cette liste en valeur et donne comme nouvelle valeur de <at> le CDR de son ancienne valeur. Cette fonction est tres utile pour avancer dans une liste qui est la valeur d'un atome.

```
(SETQ X (NEXTL L))
```

sera donc equivalente a

```
(SETQ X (CAR L) L (CDR L))
```

Cette fonction correspond en VLISP a :

```
(DF NEXTL (V) (LET ((VAR (CAR V)) (VAL (CAAR V)))
  (SET VAR (CDR VAL)) (CAR VAL)))
```

```
ex : (SETQ A '(X Y Z))    ☞  (X Y Z)
     (NEXTL A)            ☞  X
     A                    ☞  (Y Z)
     (NEXTL A)            ☞  Y
     A                    ☞  (Z)
```

(INCR <at>) [FSUBR]

<at> (qui n'est pas evalue) doit etre un atome literal dont la valeur est un nombre entier. INCR ajoute 1 a la valeur de <at> et ramene cete nouvelle valeur. Cette fonction correspond en VLISP a :

```
(SETQ <at> (ADD1 <at>))
```

```
ex : (SETQ X 2)    ☞ 2
     (INCR X)      ☞ 3
     X             ☞ 3
```

(DECR <at>) [FSUBR]

<at> (qui n'est pas evalue) doit etre un atome literal dont la valeur est un nombre entier. DECR retranche 1 a la valeur de <at> et ramene cete nouvelle valeur. Cette fonction correspond en VLISP a :

```
(SETQ <at> (SUB1 <at>))
```

```
ex : (SETQ X 3)    ☞ 3
     (DECR X)      ☞ 2
     X             ☞ 2
```

(NCONC <l1> <l2>) [SUBR a 2 arguments]

relie physiquement les deux listes <l1> et <l2> (i.e. place dans le CDR du dernier element de <l1> l'adresse de la liste <l2>). NCONC ramene la nouvelle liste <l1> en valeur.

Si <l1> et <l2> sont les memes pointeurs physiques, NCONC permet de construire des listes circulaires.

```
ex : (SETQ L1 '(A B C) L2 L1)  ☞ (A B C)
     (NCONC L1 '(D E))         ☞ (A B C D E)
     L2                        ☞ (A B C D E)
     (NCONC L1 L1)             ☞ (A B C D E A B C D E A B ...
```

(NCONC1 <l> <s>) [SUBR a 2 arguments]

est equivalent a (NCONC <l> [<s>]). Cette fonction est tres utile pour placer des elements a la fin d'une liste.

```
ex : (NCONC1 '(A B C) 'D)   ☞ (A B C D)
```

3.7 FONCTIONS SUR LES A-LISTES

En VLISP 16 comme dans tous les VLISP, les A-listes (les listes d'association) sont des listes de couples qui possedent la structure suivante :

((var1 . val1) (var2 . val2) ... (varN . valN))

Chaque element est une liste dont le CAR est une variable et le CDR une valeur.

Pour toutes les fonctions qui vont etre decrites, l'argument <al> doit etre une A-liste.

3.7.1 Recherche Sur Les A-listes

(ASSQ <a> <al>) [SUBR a 2 arguments]

ramene l'element de la A-liste <al> dont le CAR (la variable) est egal a l'atome <a>, sinon ASSQ ramene NIL. Cette fonction utilise le predicat EQ.

ex : (ASSQ 'B '((A) (B . 1) (C D E))) ☞ (B . 1)
(ASSQ 'C '((A) (B . 1) (C D E))) ☞ (C D E)

(CASSQ <a> <al>) [SUBR a 2 arguments]

est identique a ASSQ mais ramene le CDR seul de l'element de la A-liste choisi. ATTENTION : aucun moyen de faire la distinction entre la valeur NIL d'une variable et l'absence de cette variable dans la A-liste.

ex : (CASSQ 'C '((A) (B . 1) (C D E))) ☞ (D E)

3.8 FONCTIONS SUR LES P-LISTES

En VLISP 16 comme dans tous les VLISP, les P-listes (listes de proprietes) sont des listes qui ont la structure suivante :

(indic1 val1 indic2 val2 ... indicN valN)

A chaque indicateur est associe une valeur. Les recherches sur les P-listes s'effectuent donc deux elements par deux elements.

Les arguments de ces fonctions sont :

<pl> - si <pl> est un atome literal, la P-liste utilisee sera le CDR de l'atome.
- si <pl> est une liste, la P-liste utilisee sera cette liste elle-meme.

< ind> est un indicateur, atome literal ou nombre.

<pval> peut etre n'importe quelle expression.

(GET <pl> <ind>) [SUBR a 2 arguments]

ramene la valeur associee a l'indicateur <ind> dans la P-liste <pl>. Si l'indicateur n'existe pas, GET ramene NIL.
ATTENTION : on ne peut pas distinguer entre la valeur NIL associee avec un indicateur et l'absence de cet indicateur sur la P-liste.

ex : (GET '(I1 A I2 B) 'I2) ☞ B

(PUT <pl> <pval> <ind>) [SUBR a 3 arguments]

si l'indicateur <ind> existe deja sur la P-liste <pl>, sa valeur associee prend la nouvelle valeur <pval>, sinon l'indicateur <ind> et sa valeur associee <pval> sont ajoutes en queue de <pl>. PUT ramene <pl> en valeur.

ex : (PUT '(I1 A I2 B) 'C 'I1) ☞ (I1 C I2 B)

Voici la definition sous forme de memo-fonction de la fonction FIBONACCI, telle qu'aucun appel de la fonction ne sera calcule plus d'une fois, pour toute valeur de l'argument :

```
(DE FIB (N)
  (OR (GET 'FIB N)
      ((LAMBDA (X) (PUT 'FIB X N) X)
       (IF (< N 2) 1
           (+ (SELF (- N 1)) (SELF (- N 2)))))))
```

(REMP <at> <ind>) [SUBR a 2 arguments]

enleve de la P-liste <pl> l'indicateur <ind> s'il existe ainsi que sa valeur associee. REMP ramene <pl> en valeur.

```
ex : Si la P-liste de l'atome FOO est
         (I1 A I2 B)
      alors
         (REMP  'FOO 'I2)   ☞  FOO
      et
         (CDR 'FOO)         ☞  (I1 A)
      si
         (REMP  'FOO 'I1)   ☞  FOO
      alors
         (CDR 'FOO)         ☞  NIL
      enfin si
         (PUT 'FOO '(Z) 26) ☞  FOO
      alors
         (CDR 'FOO)         ☞  (26 (Z))
```

En VLISP 16, REMP peut etre definie comme

```
(DE REMP   (AT IND)
  (LET ((AT AT)) (COND
    ((NULL (CDR AT)))
    ((EQ (CADR AT) IND) (RPLACD AT (CDDDR AT)))
    (T (SELF (CDDR AT)))))
  AT)
```

CHAPITRE 4

LES NOMBRES

VLISP 16 ne met en jeu que les nombres entiers. Ils devront etre inclus dans l'intervalle [-32767 , 32767]. Cette restriction est due a la capacite mot de 16 bits du Solar.

ATTENTION, aucun test de debordement n'est effectue.

En representation decimale, un nombre negatif debute par le caractere "-", mais un nombre positif NE DOIT PAS, sauf a perdre son caractere de nombre (i.e. devenir un literal), debuter par "+" (ITA EST) .

Les nombres sont stockes dans la zone liste et sont donc geres dynamiquement.

Des nombres hexadecimaux peuvent etre utilises directement en entree, ils ont alors la representation externe :

ccccHH

les "c" devant etre des chiffres hexadecimaux.

Lors de l'impression de nombres, pour en obtenir une representation hexadecimale, evaluez (SETBIT 12) , et pour revenir a la representation decimale faites (CLRBIT 12) .

4.1 LE TEST DE TYPE

(NUMBP <s>) [SUBR a 1 argument]

Cette fonction sert a tester si l'argument <s> est ou non un nombre. Si l'argument <s> est un nombre, NUMBP ramene <s> sinon NUMBP ramene NIL.

ex : (NUMBP (ADD1 67)) ☞ 68
(NUMBP -56) ☞ -56
(NUMBP 'Nan) ☞ NIL
(NUMBP (CONS 1 2)) ☞ NIL

4.2 ARITHMETIQUE ENTIERE

Les fonctions qui vont etre decrites utilisent des operandes supposes de type entier. Ces fonctions n'effectuent aucun controle de validite de type. Si les arguments ne sont pas des nombres entiers, ces fonctions livrent en general d'horribles resultats (HORRESCO REFERENS...).

(ADD1 <n>) [SUBR a 1 argument]

ramene la valeur : <n> + 1 .

ex : (ADD1 6) ☞ 7
(ADD1 -4) ☞ -3

(1+ <n>) [SUBR a 1 argument]

identique a ADD1.

(INCR <var>) [FSUBR]

a le meme effet que

(SETQ <var> (ADD1 <var>))

(DIFFER <n1> <n2>) [SUBR a 2 arguments]

ramene la valeur : <n1> - <n2>.

ex : (DIFFER 6 -12) ☞ 18

(- <n1> <n2>) [SUBR a 2 arguments]

identique a DIFFER.

(PLUS <n1> ... <nN>) [SUBR a N arguments]

ramene la valeur : <n1> + <n2> + ... + <nN>.

ex : (PLUS 5 6) ☞ 11

```
(PLUS 5 6 4)      ☞  15
(PLUS -1)         ☞  -1
(PLUS)            ☞  0
```

(+ <n1> ... <nN>) [SUBR a N arguments]

identique a PLUS.

(QUO <n1> <n2>) [SUBR a 2 arguments]

ramene la valeur du quotient de : <n1> / <n2>.

```
ex : (QUO 20 5)     ☞  4
     (QUO 40 3)     ☞  13
```

(REM <n1> <n2>) [SUBR a 2 arguments]

ramene la valeur du reste de la division entiere de <n1> par <n2>.

```
ex : (REM 11 3)  ☞  2
     (REM 14 22) ☞  14
```

(SUB1 <n>) [SUBR a 1 argument]

ramene la valeur : <n> - 1.

```
ex : (SUB1 7)   ☞  6
     (SUB1 -9)  ☞  -10
```

A propos, que fait donc cette fonction ?

```
(DE TAK (X Y Z)
  (IF (LE X Y) Y
      (TAK (TAK (SUB1 X) Y Z)
           (TAK (SUB1 Y) Z X)
           (TAK (SUB1 Z) X Y))))
```

Pour compter le nombre de ses appels :

```
(DE CTAK (X Y Z) (LET ((C 0))
  (LET ((X X) (Y Y) (Z Z))
    (INCR C)
    (IF (LE X Y) Y
        (SELF (SELF (1- X) Y Z)
              (SELF (1- Y) Z X)
              (SELF (1- Z) X Y))))
  C))
```

TAK est-elle equivalente a NTAK ?

```
(DE NTAK (X Y Z)
  (VTAK [X Y Z]))

(DE VTAK (U) (COND
  ((NUMBP U) U)
  ((NULL (CDR U)) (1- (VTAK (CAR U))) (ADD1 'A))
  (T (LET ((X (VTAK (CAR U)))
           (Y (VTAK (CADR U))))
       (IF (LE X Y) Y
           (VTAK [[(1- X) Y (CADDR U)]
                  [(1- Y) (CADDR U) X]
                  [[(CADDR U)] X Y]]))))))
```

NTAK, redigee par John Mc CARTHY illustre la necessite absolue des listes sans type.

(1- <n>) [SUBR a 1 argument]

identique a SUB1.

(DECR <var>) [FSUBR]

a le meme effet que

(SETQ <var> (SUB1 <var>))

(TIMES <n1> ... <nN>) [SUBR a N arguments]

ramene la valeur : <n1> * <n2> * ... * <nN>.

ex : (TIMES 10 4) ☞ 40
(TIMES 2 3 4) ☞ 24
(TIMES -1) ☞ -1
(TIMES) ☞ 1

(* <n1> ... <nN>) [SUBR a N arguments]

identique a TIMES.

4.3 COMPARAISONS ENTIERES

Ces fonctions n'effectuent aucun test de validite de type. Si les arguments de ces fonctions ne sont pas des nombres, leurs resultats ne sont pas significatifs. Elles ne provoquent jamais d'erreur.

(EQ <n1> <n2>) [SUBR a 2 arguments]

ramene T si <n1> = <n2>, NIL dans le cas contraire.

(= <n1> <n2>) [SUBR a 2 arguments]

identique a EQ

(GE <n1> <n2>) [SUBR a 2 arguments]

si <n1> >= <n2> alors GE ramene <n1> sinon GE ramene NIL.

ex : (GE 3 7) ☞ NIL
(GE 4 3) ☞ 4

(GT <n1> <n2>) [SUBR a 2 arguments]

si <n1> > <n2> alors GT ramene <n1> sinon GT ramene NIL.

ex : (GT 5 5) ☞ NIL
(GT 7 4) ☞ 7

(> <n1> <n2>) [SUBR a 2 arguments]

identique a GT

(GTZ <n>) [SUBR a 1 argument]

si <n> > 0 alors GTZ ramene <n> sinon GTZ ramene NIL.

ex : (GTZ 5) ☞ 5
(GTZ 0) ☞ NIL
(GTZ -5) ☞ NIL

(LE <n1> <n2>) [SUBR a 2 arguments]

si <n1> <= <n2> alors LE ramene <n> sinon LE ramene NIL.

ex : (LE 5 5) ☞ 5
(LE 6 9) ☞ 9

(LT <n1> <n2>) [SUBR a 2 arguments]

si <n1> < <n2> alors LT ramene <n1> sinon LT ramene NIL.

ex : (LT 5 5) ☞ NIL
(LT 4 5) ☞ 4

(< <n1> <n2>) [SUBR a 2 arguments]

identique a LT

(ZEROP <n>) [SUBR a 1 argument]

ramene 0 si <n> est egal a 0 sinon ramene NIL.

ex : (ZEROP 5) ☞ NIL
(ZEROP 0) ☞ 0
(ZEROP -5) ☞ NIL

4.4 FONCTIONS LOGIQUES

En VLISP 16, un entier sur 16 bits peut etre considere comme un vecteur de bits de 16 bits. Les fonctions arithmetiques dites logiques traitent de tels vecteurs.
Pour toutes les fonctions qui vont etre decrites, l'argument s'il existe doit etre de type entier.

(LOGAND <n1> <n2>) [SUBR a 2 arguments]

Effectue l'operation de ET logique entre les deux operandes <n1> et <n2>.

ex : (LOGAND AAAAHH 03BCHH) ☞ 02A8HH

Ainsi, pour savoir rapidement si un nombre est impair

```
(DE ODDP (N) (GTZ (LOGAND 1 n)))
```

Pour savoir si un nombre est une puissance de 2

```
(DE PD2 (N) (= N (LOGAND N (DIFFER 0 N))))
```

(LOGSHIFT <n1> <n2>) [SUBR a 2 arguments]

Effectue un decalage logique de la valeur <n1>, <n2> fois. Si <n2> est positif un decalage gauche s'effectue,si <n2> est negatif, le decalage s'effectue a droite.

```
ex : (LOGSHIFT 8 3)    ☞ 64
     (LOGSHIFT 8 -2)   ☞ 2
```

(LOGOR <n1> <n2>) [SUBR a 2 arguments]

Effectue l'operation de OU logique entre les deux operandes <n1> et <n2>.

```
ex : (LOGOR 00FFHH FF00HH)  ☞ FFFFHH
```

ATTENTION, ne confondez pas les fonctions de controle AND et OR avec les fonctions arithmetiques LOGAND et LOGOR.

EXEMPLE :

Une variante de QUICKSORT :

```
(DE QS (L ;; R N Q G P S)
    ; un tri super-rapide de
      la liste de nombres L.
      FFFFFFFFFFFRRRRRRRRRRRRR........  ;
   (IF (NULL L) ()
       (SETQ R [NIL])
       (PUSH (LENGTH L) L)
       (LET ((D))
         (SETQ D (POP) N (POP))
         (IF (= N 1) D
             (SETQ Q (LOGSHIFT N -1)) (PUSH Q D)
             (SETQ D (NTH Q D))
             (PUSH (+ (LOGAND N 1) Q) (CDR D))
             (RPLACD D)
             (SETQ D (SELF) G (SELF) P (RPLACD R D))
             (WHILE D
               (IF (< (CAR G) (CAR D))
                   (SETQ N D D G G N N (RPLACD P D)))
               (SETQ P D D (CDR D)))
             (RPLACD P G) (CDR R)))))
```

avec les macros

```
(DM PUSH (CALL) (RPLACB CALL
  ['SETQ 'S (CONS 'MCONS (REVERSE (CDR CALL) ['S]))]))

(DM POP (CALL) (RPLACB CALL '(NEXTL S)))
```

CHAPITRE 5

ENTREES SORTIES ET FICHIERS

Des expressions VLISP peuvent etres lues et ecrites sur les terminaux ou le disque (D3 ou D7 a l'Ecole).

Il est bon de savoir, en ce qui concerne les entrees-sorties sur terminaux, que le LISP que vous tapez est lu sur CC, que le LISP imprime l'est sur LL, que le line-feed gratuit imprime en fin de ligne (quand vous frappez RETURN) l'est sur EL, ainsi que les messages d'erreur et le prompteur "?".

La fin d'une ligne est consideree par le lecteur VLISP comme un separateur au meme titre que le caractere ESPACE (un atome VLISP, nombre ou symbole ne doit donc pas etre a cheval sur 2 lignes).

Pour annuler les n derniers caracteres tapes sur une ligne de terminal, tapez n fois le caractere " \ " (antislash) puis continuez votre entree sur la ligne (une ligne s'acheve par la frappe d'un RETURN).
Pour annuler toute une ligne d'un coup, tapez "\" suivi immediatement de RETURN.

ATTENTION, ne depassez pas 76 caracteres par ligne, sinon il se produira des evenements consternants (cet avertissement vaut egalement pour les lignes de fichiers prepares avec EDIT16).

En entree, tant sur terminal que sur un fichier, tout ce qui se trouve entre deux point-virgules (" ; ") est TOTALEMENT ignore de VLISP. Vous pouvez ainsi inserer des commentaires (meme vides) dans et entre vos expressions.

Enfin, VLISP 16 effectue de lui-meme des indentations en debut de votre ligne de terminal (apres le prompt " ? ") dont le nombre (un multiple de 3) est proportionnel a la profondeur d'imbrication de votre expression parenthesee. Cette indentation demeure stable au dela de la profondeur 7.

Cette douceur a ete inventee par Jerome CHAILLOUX.

Si vous desirez vous en passer, chantez (SETBIT 3).

Tous les fichiers d'entree sont compatibles avec l'editeur EDIT16 .

5.1 LES SPECIFICATIONS DE FICHIERS

Une specification de fichier a en VLISP 16 la forme suivante :

<filnam><ct> OU <filnam>:L

dans laquelle :

<filnam> : est le nom du fichier (6 caracteres exactement).
<ct> : est le nom de catalogue (2 caracteres exactement).

Le catalogue ":L" est reserve pour les fichiers standard de VLISP 16 (ex : PRETTY:L, TRACEF:L, ...).

SUPER ATTENTION !!! :

Sous TSF et EDIT16 le nom du fichier comporte un tiret ("-") entre le nom proprement dit et le catalogue de 2 lettres qui le suit.

Omettez imperativement ce tiret sous VLISP 16. Les sceptiques ou les distraits seront irremediablement perdants.

Je repete. PAS de tiret entre le <filnam> et le <ct> en VLISP 16.

5.2 LA SELECTION DES FICHIERS D'ENTREE/SORTIE

VLISP 16 donne acces a la plupart des fonctions de FMS (le fameux systeme de fichiers du SOLAR).

Les fichiers standard sont des fichiers SEQUENTIELS.

Neuf fichiers au maximum peuvent etre simultanement ouverts. Chacun d'entre eux sera associe a un canal logique <nc>. Apres ouverture, seul le <nc> permet d'y acceder.

(FMS <file> <nf> <nc>) [SUBR a 3 arguments]

avec <file> : une specification de fichier, <nf> : un "numero-de-fonction", et <nc> : un "numero-de-canal-logique" au sens de FMS.

<nc> doit etre compris entre 1 et 9 inclus.

<nf> peut etre :

1 : OPEN NEW , ouvre un NOUVEAU fichier temporaire.
2 : OPEN OLD , ouvre un fichier DEJA-EXISTANT.
3 : CLOSE , ferme le fichier ouvert associe au canal logique <nc>.
4 : CREATE , cree et ouvre un NOUVEAU fichier permanent.
5 : CATAL , rend permanent le fichier temporaire associe a <nc>.
6 : DELETE , tue le fichier ouvert associe au canal <nc>.
9 : RENAME , rebaptise <file> le fichier ouvert associe a <nc>.
10: EOJ , ferme absolument TOUS les fichiers ouverts.

Dans le cas ou vous tentez d'ouvrir un fichier inexistant, ou deja ouvert et non referme, FMS ramene la valeur NIL.
Si l'ouverture est correcte, FMS ramene le nom du fichier en valeur.

(FMSS <nf> <nc>) [SUBR a 2 arguments]

ne sert que pour certains positionnements. Accomplit sur le fichier associe au canal <nc> les operations suivantes :

126 : REWIND . Toute lecture ou ecriture se fera en DEBUT de fichier.
127 : SKEOA . Toute ecriture se fera A LA SUITE du fichier existant.

(INPUT <ip>) [SUBR a 1 argument]

avec <ip> pouvant etre un canal logique <nc>, ou bien NIL.

Si <ip> est un <nc> alors toutes les lectures (par READ, READCH ...) se feront sur le fichier ouvert associe, sur disque.
Si <ip> est NIL, toutes les lectures se feront sur le peripherique (clavier, cartes ...) precedant la demande de lecture sur disque.

(OUTPUT <op>) [SUBR a 1 argument]

avec <op> pouvant etre un canal logique <nc>, ou bien NIL.

Si <op> est un <nc> alors toutes les ecritures (par PRINT, TERPRI ...) se feront sur le fichier ouvert associe, sur disque.
Si <op> est NIL, toutes les ecritures se feront sur le peripherique (terminal, imprimante ...) precedant la demande d'ecriture sur disque.

EXEMPLE :

Une fonction pour lire sur disque une suite d'expressions, jusqu'a rencontre de l'atome EOF, et en faire une liste.

```
(DE FOO (NOMFIC)
  (FMS NOMFIC 2 1)        ; ouvrir le fichier deja existant ;
                          ; en l'associant au canal 1      ;
  (INPUT 1)               ; preparer la lecture sur canal 1 ;
  (LET ((X (READ)))
    (IF (NEQ X 'EOF) (CONS X (SELF (READ)))
        (FMS () 3 1)      ; fermer le fichier               ;
        (INPUT)           ; revenir a la lecture normale    ;
        NIL)))
```

(STATUS 8 <nfu>) [SUBR a 2 arguments]

avec <nfu> : un numero de FU-disque. Permet de changer de FU-disque, tant pour les ecritures que pour les lectures. <nfu> devient la FU courante.

Il est bon de savoir que (en decimal) :

D2	D3	D4	D5	D6	D7	D8
14	15	16	17	18	19	20

5.3 LES FONCTIONS D'ENTREE DE BASE

Elles utilisent un buffer d'entree inaccessible a l'utilisateur et lisent sur le fichier d'entree choisi au moyen de la fonction INPUT.

Les elements de la ligne sont pris en compte par le lecteur VLISP 16 lorsque la fin de l'enregistrement est specifiee (sauf dans le cas de (STATUS 5)) par la frappe d'un <cr> i.e. RETURN.

On notera qu'en lecture, des symboles atomiques comportant des caracteres separateurs (parentheses, point, espace ...) peuvent etre aisement crees en faisant preceder ces caracteres problematiques de " / " qui leur donne un statut de caractere ordinaire.

ex : AN/(NIE/)
sera lu comme l'atome de 7 caracteres
AN(NIE)

Il va de soi que le "/" peut se quoter lui-meme.

On notera egalement avec plaisir qu'une lecture au top-level peut debuter par un nombre quelconque de parentheses fermantes qui sont ignorees. Ceci a la plaisante consequence de permettre de refermer a coup sur les definitions de fonctions par une forte giclee de ")", sans avoir a les denombrer (UTI, NON ABUTI) .

Enfin on notera qu'au terminal, la frappe de la touche ESC vous fait sortir de VLISP sur l'instant (de la meme facon, en moins lourd qu'un (RESET T)) et fait revenir sous TSF.

(READ) [SUBR a 0 argument]

lit un objet (symbole atomique, nombre ou liste) dans le buffer d'entree, et ramene cet objet en valeur.

(READCH) [SUBR a 0 argument]

lit le premier caractere immediatement disponible dans le buffer d'entree et le ramene en valeur sous la forme d'un atome cree ou reconnu. Si la fin du buffer d'entree est atteinte, un nouvel enregistrement est alors automatiquement lu.

EXEMPLE :

```
?[(READCH) (READCH) (READCH)]FOO
 (F O O)
```

(STATUS 5) [SUBR a 1 argument]

lit de meme que READCH un caractere , mais ce caractere est transmis et devient testable meme si l'enregistrement d'entree n'est pas complet. Au terminal, ceci signifie que le caractere est lu par (STATUS 5) dans un buffer separe, et peut etre traite immediatement apres avoir ete tape sans etre suivi de <cr> i.e. RETURN.

ex :
```
? (IF (EQ 'a (STATUS 5))'OK 'KO)
?a OK
```

(STATUS 6) [SUBR a 1 argument]

identique a READCH mais n'avance pas dans le buffer, le caractere lu demeure disponible pour une prochaine lecture.

ex :
```
? (PROGN (PRINT (STATUS 6)) (PRINT (READCH))) a
a
a
```

tres bon pour avoir un caractere d'avance.

5.4 LE MODE LIBRARY

Un nouveau mode de chargement rapide a ete ajoute : le mode LIBRARY, qui, silencieusement, vous charge n'importe quel fichier de fonctions se terminant obligatoirement par l'expression (INPUT).

ATTENTION : les fonctions se trouvant dans ce type de fichier ne doivent pas contenir d'erreurs de syntaxe (de lecture), ce mode est destine principalement aux chargements rapides des fichiers systemes et des fichiers utilisateurs ne provoquant pas d'erreurs de lecture.

(LIBRARY <file>) [FSUBR]

charge le fichier <file> en silence (sans aucune impression). Si le fichier existe et si le chargement s'est correctement effectue, cette fonction ramene (<file>) en valeur i.e. le nom du fichier charge. Dans tous les autres cas LIBRARY ramene NIL.

ATTENTION : Le fichier charge par LIBRARY doit imperativement se terminer par (INPUT).

5.5 LE MODE AUTOLOAD.

Pour les fichiers contenants des ensembles de fonctions tres souvent utilisees il serait assommant de devoir les charger d'abord par un appel de LIBRARY, puis d'appeler la fonction principale du fichier ainsi charge. Le mode dit AUTOLOAD resout heureusement cette difficulte.

(AUTOLOAD <file>) [SUBR a 1 argument]

rend le fichier <file> autoloadable. Supposons que FOO est une des fonctions du fichier FOOFIC:L. Si vous evaluez :

```
(DM FOO (L) (AUTOLOAD 'FOOFIC:L))
```

Si a present vous appelez (FOO ...), le fichier FOOFIC:L sera automatiquement charge (il doit naturellement se terminer par (INPUT) comme tous les fichiers de fonctions), puis la fonction FOO sera appelee avec sa definition chargee et ses arguments originaux. C'est ce qui vous permet de chanter directement des choses comme (PRETTY HACK) ou (TRACE HACK) sans avoir a charger vous-memes PRETTY:L ou TRACEF:L par un appel de LIBRARY.

5.6 LES FONCTIONS DE SORTIE DE BASE

Toutes editent dans un buffer de sortie totalement accessible a l'utilisateur, et ecrivent sur le fichier de sortie choisi au moyen de la fonction OUTPUT. Si au cours d'une edition le buffer de sortie est plein, il est automatiquement imprime, en utilisant la fonction (TERPRI), et l'edition se poursuit sur la ligne suivante. Sauf modification demandee (voir Chapitre 6) tous les objets edites par PRINT et PRIN1 seront precedes d'un espace.

(PRINT <s1> ... <sN>) [SUBR a N arguments]

edite dans le buffer de sortie les differentes S-expressions <s1> ... <sN> puis imprime ce buffer (apres insertion des caracteres Carriage Return / Line Feed). PRINT ramene <sN> en valeur.

(PRIN1 <s1> ... <sN>) [SUBR a N arguments]

edite dans le buffer de sortie les differentes S-expressions <s1> ... <sN> sans imprimer le buffer. PRIN1 ramene <sN> en valeur.

(TERPRI) [SUBR a 0 argument]

imprime le buffer de sortie, se positionne en debut de ligne en imprimant le code Return, puis saute une ligne en imprimant une fois le code Line-feed.

TERPRI ramene toujours la valeur NIL.

ex : La fonction PR-ATS imprime les atomes elements de la liste L dans leur ordre d'occurence au cours du balayage prefixe :

```
(DE PR-ATS (L)
  (LET ((L L)) (COND
    ((NULL L))
    ((ATOM L) (PRIN1 L))
    ((ATOM (CAR L)) (PRIN1 (NEXTL L)) (SELF L))
    (T (SELF (NEXTL L)) (SELF L))))
   (TERPRI))

?(PR-ATS '(A (B . C) NIL (D E (F))))
 A B C NIL D E F
 NIL
```

(PAGE <n>) [SUBR a 1 argument]

imprime le buffer de sortie puis saute <n> pages (en envoyant le code Form-Feed) sur LO. PAGE ramene <n> en valeur.

(SPACES <n>) [SUBR a 1 argument]

edite <n> fois le caractere espace. SPACES ramene <n> en valeur.

(TTAB <n>) [SUBR a 1 argument]

specifie que le prochain objet edite (par PRINT ou PRIN1) sera place a la position <n> dans la ligne de sortie. TTAB ramene <n> en valeur. TTAB est utilise pour creer des tabulations (alignages sur des colonnes specifiees) programmees.

5.7 MACROS-CARACTERES

Il y a possibilite A LA LECTURE d'appeler une fonction VLISP a la seule occurence d'un caractere dans le flot d'entree. Le caractere associe prend alors le statut de macro-caractere.

Le resultat de l'appel de cette fonction sera substitue a la place du caractere lu. Cette possibilite est tres utilisee par VLISP lui-meme (le caractere de quotage "'" est un macro-caractere), ainsi que dans les programmes ou il y a lieu de distinguer les variables des objets constants : interpretes speciaux, unifications, filtrages. Plus generalement un objet, atome ou liste, precede d'un macro-caractere pourra etre transforme au moment de la lecture en une expression denotant explicitement son type. Des exemples viendront eclairer cette explication (!) tres obscure.

(MCHAR <c> <lam>) [FSUBR]

ou <c> est le caractere a macrocariser, et <lam> est une lambda-expression.

Associe le caractere a la lambda-expression qui se trouvera appelee avec 0 argument, aux occurences du caractere dans le flot d'entree.

On notera qu'un macro-caractere se comporte comme un caractere normal s'il est precede de "/".

EXEMPLES:

Si le macro-caractere QUOTE "'" n'etait pas standard, il serait defini par :

```
(MCHAR /' (LAMBDA () (LIST QUOTE (READ))))
```

Pour que [e1 e2 ... eN] soit, a la lecture, transforme en (LIST e1 e2 ... eN) :

```
(MCHAR /[ (LAMBDA () (LET ((X 'LIST))
        (IF (NEQ X '/]) (CONS X (SELF (READ)))))))

(MCHAR /] (LAMBDA () '/]))
```

Le plus simple des filtrages :

```
(MCHAR ! (LAMBDA () ['VAR (READ)]))

(DE MATCH (P E)
  (ESCAPE EXIT
    (LET ((P P) (E E) (AL)) (COND
      ((ATOM P) (IF (NEQ P E) (EXIT 'NO)
                AL))
      ((EQ (CAR P) 'VAR)
       (LET ((Z (ASSQ (CADR P) AL)))(COND
         ((NULL Z) (CONS [(CADR P) E] AL))
         ((EQUAL (CADR Z) E) AL)
         (T (EXIT 'NO)))))
      (T (SELF (CDR P) (CDR E)
               (SELF (CAR P) (CAR E) AL)))))))


?(MATCH '(!X * !X) '((A - B) * (A - B)))
 ((X (A - B)))

?(MATCH '(!X * (!Y + !Z)) '((A + B) * (C + D)))
 ((Z D) (Y C) (X (A + B)))

?(MATCH '(!X * !X) '(5 * A))
 NO
```

CHAPITRE 6

LE MOT DE CONTROLE

Il existe un mot de 16 bits qui controle certains etats du systeme VLISP 16. Ce mot est accesible au moyen des fonctions SETBIT et CLRBIT.

6.1 LES FONCTIONS DU MOT DE CONTROLE

(SETBIT <n>) [SUBR a 1 argument]

positionne a 1 le bit de numero <n> du mot de controle. <n> doit etre compris entre 0 et 15 inclus. Ramene <n> en valeur.

(CLRBIT <n>) [SUBR a 1 argument]

identique a SETBIT mais positionne a 0 le bit <n> du mot de controle.

Voici la liste de ces bits ainsi que leur signification.

BITS	FONCTION DU POSITIONNEMENT
15	= 0 : un espace est place AVANT tout objet, atome ou liste, imprime dans une ligne. = 1 : suppression de cet espace.
14	= 0 : sortie teletype. = 1 : sortie sur LO (l'imprimante). En BOS/D uniquement.
13	= 0 : les macro-caracteres sont pris en compte. = 1 : il n'en est rien. Les caracteres definis comme macro-caracteres sont lus comme des caracteres ordinaires.
12	= 0 : sortie des nombres en decimal. = 1 : sortie des memes en hexadecimal.
11	= 0 : les ";" sont pris en compte comme separateurs de commentaires. = 1 : ils ne le sont pas. ";" est alors un caractere ordinaire.
10	= 0 : les "/" sont pris en compte comme quoteurs de caracteres. = 1 : ils ne le sont pas. "/" est alors un caractere ordinaire.
9	= 0 : ne pas restituer les "/" en sortie = 1 : les restituer en sortie devant " ", "'", "/", ".", "(", ")" .
8	= 0 : impression de la derniere evaluation demandee au TOP-LEVEL. = 1 : non-impression au TOP-LEVEL.

7	= 0 : listing carte lue sur LO. En general l'imprimante. = 1 : listing carte lue sur teletype. Seulement sous BOS/D.
6	= 0 : rien = 1 : lecture sur disque. Ne le positionnez JAMAIS vous-meme.
5	= 0 : rien = 1 : ecriture sur disque. N'y touchez PAS non plus. Des choses horribles pourraient arriver.
3	= 0 : l'entree-clavier est indentee. = 1 : elle ne l'est pas (pour que les conservateurs puissent conserver).
2	= 0 : non-listing d'enregistrement lu sur SI (lecteur de cartes ou disque). = 1 : listing du dit enregistrement.
1	= 0 : mode interactif. seul valable sous TSF. = 1 : mode BATCH (sous BOS/D seulement).
0	= 0 : lecture sur teletype. = 1 : lecture sur SI (le lecteur de carte).

Au chargement de VLISP 16, tous ces bits sont a 0.
Toute erreur remet de meme le mot de controle a 0; ainsi, les options standard sont restaurees.

CHAPITRE 7

ERREURS, STATUS, ET FONCTIONS SYSTEME

7.1 DIAGNOSTICS D'ERREUR

S'il y a une erreur detectee, VLISP tape "*ER <le-type-d'-erreur>" precede dans certains types de l'impression du coupable (ex: la fonction non-definie), puis repasse la main au clavier et a la boucle de lecture de l'interprete.

Aucune definition de fonction, ou liaison de variable n'est modifiee, et un garbage-collecting est effectue dans la foulee.

Voici la liste des <types-d'-erreur> :

LC : erreur de lecture. Il peut s'agir d'un "." mal place, ou d'un "[" non-referme. On est amene souvent a la provoquer volontairement.

RT : on a tente d'evaluer un appel de RETURN sans etre dans un PROG.

FS : pile de travail debordee. Erreur due a une recursion trop profonde ou infinie.

FM : il n'y a plus de doublets disponibles en zone liste.

AT : il y a trop d'atomes definis par l'utilisateur.

A2 : fonction non-definie, detectee dans APPLY. Le nom de cette fonction inconnue est imprime AVANT le diagnostic.

A6 : GO ou GOTO a une etiquette absente du PROG courant. Le nom de l'etiquette absente est imprime.

A8 : tentative d'evaluation d'une variable non-initialisee. Le nom de la variable est imprime. En general, la variable a ete simplement oubliee.

A9 : fonction non-definie, detectee dans EVAL. Le nom de la fonction inconnue est imprime.

7.2 LES DESASTRES

En dehors des erreurs civilisees attrapees et signalees par VLISP 16, des situations consternantes peuvent se produire. Depuis l'absence totale de reponse du terminal, un comportement de sortie erratique, jusqu'aux erreurs brutales FMS vous renvoyant sous TSF.

Certains de ces desastres sont comprehensibles. En voici une breve liste accompagnee d'un remede quand ya moyen. Ya des cas ou ya pas de remede. Tant pis.

- Un crash TSF (Dieu sauve vos fichiers !).
- Vous bouclez (je ne peux rien faire pour vous).
- Vous avez lu une ligne de plus de 76 carcateres. Coupez la en 2 lignes de longueur canonique.
- Vous avez fait un SETBIT inepte. Relisez le Chapitre 6.
- Vous avez frappe la touche BREAK en plein milieu d'un garbage-collecting (je n'y puis rien. Je ne peux pas masquer les interruptions pendant le garbage-collecting sous TSF en mode esclave).
- Vous avez modifie une constante. L'effet peut etre des plus curieux, specialement s'il s'agit de T. N'utilisez pas T comme variable, ni aucune constante d'ailleurs.
- Vous avez effectue un RPLACA ou un RPLACD sur un nombre. Effet prodigieux. Trouvez ou ca se produit et corrigez.
- Vous appliquez des fonctions arithmetiques sur des objets qui ne sont pas des nombres, ou l'inverse. Tres sournois. Corrigez.
- Vous avez fait un appel avec NIL en position de fonction. Ca boucle. Pas facile a trouver, mais on y arrive quand meme.
- Vous avez INCONSIDEREMENT fait un appel avec une constante en position de fonction. Le resultat peut etre surprenant.
- Il manque des parentheses fermantes dans votre fichier de fonctions. Rajoutez-les aux bons endroits.
- Votre fichier de fonctions ne se termine pas par (INPUT). Mettez-le.
- Vous avez mis le tiret fatidique (voir Chapitre 5) dans votre nom de fichier entre <filnam> et <ct>. Enlevez-le.
- Votre fichier n'existe pas. Creez-le.

7.3 STATUS

La fonction STATUS sert a accomplir des actions hardies, telles que la modification physique de l'interprete en memoire. Dieu vous garde et fasse que vos errances n'aboutissent pas a la destruction totale de votre image-memoire.

(STATUS 1 <n>) [SUBR a 2 arguments]

ramene le contenu du mot memoire d'adresse <n> (adresse absolue sous BOS/D, relative sous TSF).

(STATUS 2 <n1> <n2>) [SUBR a 3 arguments]

place en memoire le nombre <n2> a l'adresse <n1>. Dieu vous garde etc.

(STATUS 3) [SUBR a 1 argument]

sauve l'ensemble des atomes que vous avez definis jusqu'ici.

(STATUS 4) [SUBR a 1 argument]

restitue l'ensemble des atomes sauves par (STATUS 3). Permet donc de lutter contre la penurie chronique de place pour les atomes. Apres un (STATUS 4) tous les atomes definis depuis le dernier (STATUS 3) sont volatilises. Utilise en general par les programmes a grand nombre d'atomes tels que LAP ainsi que par les systemes de traitement de langage naturel.

(STATUS 5) (STATUS 6) (STATUS 8 <n>)

voir le chapitre Entrees-Sorties en 5.3 et 5.2 .

(STATUS 6 <n>) voir en 3.5 .

7.4 FONCTIONS SYSTEME

En VLISP 16, un nombre est un doublet en zone liste de la forme :

(<marque-de-nombre> . <le-nombre-16-bits>)

Pour des actions operables par les seuls magiciens, la possibilite de passer de la representation <nombre-VLISP> a la representation <nombre-16-bits> est disponible.

(LOC <s>) [SUBR a 1 argument]

donne, sous la forme d'un <nombre-VLISP> l'adresse en memoire de l'expression <s>, symbole, nombre ou liste.

EX :

(LOC NIL) ☞ adresse de l'atome NIL.

(VAG <n>) [SUBR a 1 argument]

remene l'objet VLISP, nombre, symbole ou liste, habitant a l'adresse <n>.

EX :

(VAG (LOC expression)) ☞ l'expression

(RESET T) [SUBR a 1 argument]

redonne la main au systeme (BOS/D ou TSF). N'est plus guere utilise qu'en mode BATCH. En mode interactif, on sort de VLISP par la frappe de ESCAPE au clavier. Pour rentrer a nouveau dans VLISP, toutes definitions et liaisons conservees, chanter a TSF :

>CLISP

LOC et VAG seront tres utiles pour s'allouer des doublets en nombre superieur a celui donne par defaut.

EXEMPLE :

Pour obtenir 300 doublets supplementaires, chantez au TOP-LEVEL

```
(PROGN (SET (CDR 'CAAR) (VAG (+ 600 (LOC (CADR 'CAAR)))))
       (SET (CDR 'CADR) (VAG (+ 600 (LOC (CADR 'CADR)))))
       (XYZZY))
```

Et vous voila avec 300 doublets additionnels.

CHAPITRE 8

EDITIONS ET TRACES

Ce chapitre est consacre a divers outils d'impression, de trace, d'edition, et de mise au point. En VLISP comme ailleurs, on vit tres mal sans.

8.1 LE PRETTY-PRINT

Le Pretty-Print est un ensemble de fonctions VLISP destine a l'impression supposee agreable et lisible de fonctions presentes en memoire. L'impression obtenue est essentiellement une justification selective par indentation.

Pour l'utiliser il suffit de chanter :

(PRETTY <nom1> <nom2> ... <nomN>)

et voici que sont prettyprintees les fonctions <nom1> ... <nomN>.

8.2 LES TRACES

Il s'agit encore d'un ensemble de fonctions VLISP destine a imprimer des etats et resultats intermediaires au cours d'un calcul. Plusieurs types de traces sont disponibles. Dans tous les cas l'unite de trace est la fonction. Dans ce qui suit, tous les <nom1> ... <nomN> designeront des noms de fonctions definies par l'utilisateur, de type EXPR ou FEXPR.

(TRACE <nom1> ... <nomN>) [FEXPR]

Tous les appels des fonctions nommees seront imprimes ainsi que leur niveau d'appel et la liste des valeurs des arguments. Au retour sera imprime le nom de la fonction, le niveau de retour ainsi que le resultat de l'appel correspondant.

(UNTRACE <nom1> ... <nomN>) [FEXPR]

Annule les demandes de trace pour les fonctions nommees. Je ne vous conseille pas d'essayer de untracer des fonctions non tracees ; le resultat sera horrifiant.

(TRACEQ <nom1> ... <nomN>) [FEXPR]

Dans les fonctions nommees, toutes les affectations par SETQ (seulement dans le cas ou le SETQ n'affecte qu'une variable) seront tracees sous la forme :

<variable> = <valeur-affectee>

(UNTRACQ <nom1> ... <nomN>) [FEXPR]

Annule les demandes de traces de variables affectees pour les fonctions nommees.

(TRACEGO <nom1> ... <nomN>) [FEXPR]

Lorsque les fonctions nommees contiennent un PROG, TRACEGO permet de tracer tous les branchements par GO. Sera, a chaque branchement effectue, imprime :

(ETIQ: <etiquette-de-branchement>)

(UNTRACG <nom1> ... <nomN>) [FEXPR]

Annule les demandes de trace de branchements pour les fonctions nommees.

L'ensemble de ces fonctions de trace est rassemble dans le fichier TRACEF-:L. Examinez-le donc avec EDIT16.

8.3 L'EDITEUR EF, ADVISE ET BREAK.

EF est un petit editeur, tres rapide, pas encombrant et pratiquement incassable.

Il a l'avantage considerable de n'etre PAS une boucle TOP-LEVEL, et donc de donner acces a l'unique structure de donnee de EF : la variable globale "**" (voir listing au chapitre 9), et de permettre a tout instant l'evaluation de la fonction en cours d'edition.

Pour l'utiliser il convient de charger par LIBRARY le fichier DEBUGG-:L.

Une fois qu'on est entre dans EF, et nous verrons comment, il faut s'imaginer qu'un CURSEUR est positionne a gauche du premier element de ce que nous nommerons dans ce qui suit la LISTE COURANTE.

EXEMPLE :
Voici une fonction problematique :

```
(DE FOO (X Y)
    (FI (NULL Z)
        (CONS (FOO (CAR X)) (FOO CDR X))))
```

Pour l'editer on chante :

```
(EF FOO)
```

Et EF replique :

```
(LAMBDA (* *) (* * *))
```

Les etoiles remplacent (a l'impression seulement) les niveaux depassant la profondeur 1.

Et voici ce qu'il faut s'imaginer :

```
Le-LISTE-COURANTE--->( LAMBDA (* *) (* * *))
                     ↑
                     Le-CURSEUR
```

L'edition consistera a deplacer le curseur et a modifier les zones insatisfaisantes, et la suite va nous indiquer comment.

8.3.1 Commandes De EF

(EF <nom-de-fonction>) [FEXPR] {Edit Function}

Pour rentrer sous editeur avec la fonction.

(P <n>) [EXPR a 1 argument] {Print}

Pour faire imprimer l'objet edite a <n> niveaux de profondeur (les niveaux trop profonds sont imprimes "*").

8.3.1.1 Recherche Par Position -

(MV <q1> ... <qN>) [FEXPR] {MoVe}

Deplace le curseur comme suit. Si <qI> est un nombre, le curseur est deplace a gauche du qI-eme element de la LISTE COURANTE. Si <qI> est l'atome "UP", l'element sur lequel pointe le curseur doit etre une liste, cette liste devient alors la LISTE COURANTE.

<qI> ::= <n> | UP

EXEMPLES :

```
liste-courante : (A (B) (C D E))
commande       : (MV 3 UP 1)
liste-courante : (C D E)

liste-courante : (IF (FOO X) (BAR (CAR X Y)))
commande       : (MV 3 UP 2 UP 2)
liste-courante : (X Y)
```

8.3.1.2 Recherche Par Contenu -

(FK <s>) [FEXPR] {Find Konstant}

Avec <s> une expression, atome ou liste. FK considere la liste courante comme un arbre et y effectue une recherche prefixe jusqu'a trouvaille de <s>, a la suite de quoi la recherche cesse, en positionnant le curseur a gauche de l'objet trouve.

EXEMPLES :

```
liste-courante : (A (B C) (B E))
commande       : (FK B)
liste-courante : (B C)

liste-courante : (IF (NULL X) NIL (CONS (CAR X) (FOO (CDR X))))
commande       : (FK (CAR X))
liste-courante : ((CAR X) (FOO (CDR X)))
```

(FP <filtre>) [FEXPR] {Find Pattern}

Le <filtre> est :
- une liste ordinaire OU
- une liste comportant des occurences de "?" OU
- une liste pointee (e1 e2 ... eN . ?) .

FP balaye de meme la liste courante comme un arbre prefixe a la recherche d'une sous-expression heureusement filtrable et s'arrete avec positionnement lorsque l'heureux evenement se produit.

EXEMPLES :

```
liste-courante : (IF (NULL X) NIL (CONS (CAR X) (FOO (CDR X))))
commande       : (FP (NULL ?))
liste-courante : ((NULL X) NIL (CONS (CAR X) (FOO (CDR X))))
commande       : (FP (FOO . ?))
liste-courante : ((FOO (CDR X)))
```

8.3.1.3 Commandes De Modification -

(I <e1> <e2> ... <eN>) [FEXPR] {Insert}

Insere a DROITE du curseur les expressions <e1> ... <eN>.

(IL <e1> <e2> ... <eN>) [FEXPR] {Insert Last}

Insere a la FIN de la liste courante les expressions <e1> ... <eN>.

(D <n>) [EXPR a 1 argument] {Delete}

Elimine les <n> expressions a droite du curseur.

(DL) [FEXPR] {Delete Last}

Elimine la DERNIERE expression de la liste courante.

C'est tout. C'est simple, pratique et agreable.

8.3.2 ADVISE

Permet d'interposer des couches d'instructions AVANT l'entree dans le corps d'une fonction, et APRES l'evaluation du corps. Cette interposition est faite de facon totalement independante de la definition de la fonction ADVISEe, et ne change en rien le resultat que la fonction est supposee retourner.

(ADVISE <nom-de-fonction> <e1> ... <eN> * <d1> ... <dM>) [FEXPR]

ADVISE insere les instructions <e1> ... <eN> AVANT le corps de la fonction (represente obligatoirement par "*"), et insere les instructions <d1> ... <dM> APRES le corps de la fonction.

EXEMPLE :

```
(DE FOO (X) (PLUS X 4))
(FOO 3)     ☞ 7

(ADVISE FOO (PRINT 'BONJOUR) * (PRINT 'BYE))
(FOO 3)     ☞ BONJOUR
              BYE
              et 7 est toujours ramene en valeur.
```

(UNADVISE <nom-de-fonction>) [FEXPR]

De-ADVISE la fonction precedemment ADVISEe.

EXEMPLE : (suite du precedent)

```
(UNADVISE FOO)
(FOO 3)    ☞ 7
```

8.3.3 BREAK

BREAK permet de placer une boucle de lecture-evaluation-ecriture a l'entree d'une fonction, de facon tout a fait independante de la definition de la fonction. En mode interactif, BREAK ressemble a une trace dans laquelle, a l'entree dans la fonction, des informations sur les valeurs de variables, des evaluations, voire des editions peuvent etre demandees. Tres utile outil de mise au point.

(BREAK <nom-de-fonction>) [FEXPR]

Induit l'effet complexe suivant :
apres cette commande, toutes les fois que la fonction BREAKee sera appelee, vous vous retrouvez dans une boucle de TOP-LEVEL qui evalue et imprime ce que vous tapez (toutes sortes de demandes de renseignements pourquoi votre fonction ne marche pas, je suppose), jusqu'a tapage d'un "T", a la suite de quoi la fonction se deroule normalement.
La boucle TOP-LEVEL en question est :

```
(DE TOP (MSG ;; -X-)
    (PRINT MSG)
    (IF (EQ (SETQ -X- (READ)) T)
        T
        (TOP MSG)))
```

Si elle ne vous plait pas, redefinissez TOP a loisir, et avec mes compliments.

(UNBREAK <nom-de-fonction>) [FEXPR]

DeBREAKe la fonction BREAKee.

Le listing de EF, BREAK et ADVISE est fourni au chapitre 9.

CHAPITRE 9

QUELQUES EXEMPLES

Dans ce chapitre, le lecteur trouvera quelques exemples d'utilisation de VLISP 16. Des techniques tres variees y sont mises en jeu, illustrant divers aspects du langage et de son utilisation ordinaire. Tous ces programmes sont essentiellement didactiques et visent moins a instruire le lecteur qu'a l'inciter a etendre les portees d'action de ces programmes consideres comme noyaux de developpement. Reste qu'on ne trouvera pas ici pour chacun d'eux un commentaire detaille, ce n'est pas l'objet d'un manuel de reference. Seront livres cependant une rapide description de leurs raisons, des exemples d'utilisations, le listing complet enfin. Faites donc tourner tout ca pour votre edification et votre plaisir.

9.1 MINI-SERIES

Un programme de completion de series ecrit a l'origine en langage SAIL par C. HEDRICK a C.M.U., en Janvier 1972. Le programme tente de former une description d'une serie de nombres, et s'efforce d'utiliser cette description pour etendre les series decrites.

Une description sera une suite de triplets :

(<start> <move> <increment>)

Un triplet est une sorte de peigne a 3 branches. La premiere pointe sur une position de depart dans la serie, la seconde indique la distance entre <start> et <move>, la troisieme designe l'increment a ajouter aux nombres pointes par les branches.

Voici des exemples d'utilisation de MINSER.

```
?(M-SERIES '(1 1 1 1 ? ? ?))
 SUCCES (1 1 1 1 1 1 1)
 3-UPLES ((1 1 0))
 OK

?(M-SERIES '(1 2 3 4 ? ?))
 SUCCES (1 2 3 4 5 6)
 3-UPLES ((1 1 1))
 OK

?(M-SERIES '(1 2 1 2 1 ? ? ?))
 ECHEC (1 2 1 2 1 ? 1 ?)
 3-UPLES ((1 2 0))
 OK

?(M-SERIES '(1 2 1 2 1 2 ? ? ?))
 SUCCES (1 2 1 2 1 2 1 2 1)
 3-UPLES ((2 2 0) (1 2 0))
 OK

?(M-SERIES '(1 3 2 4 3 5 ? ? ? ?))
 SUCCES (1 3 2 4 3 5 4 6 5 7)
 3-UPLES ((2 2 1) (1 2 1))
 OK

?(M-SERIES '(1 10 2 9 3 8 4 ? ? ? ?))
 SUCCES (1 10 2 9 3 8 4 7 5 6 6)
 3-UPLES ((2 2 -1) (1 2 1))
 OK

?(M-SERIES '(8 6 4 2 ? ? ?))
 SUCCES (8 6 4 2 0 -2 -4)
 3-UPLES ((1 1 -2))
 OK

?(M-SERIES '(1 1 2 3 3 5 4 7 ? ? ? ?))
 SUCCES (1 1 2 3 3 5 4 7 5 9 6 11)
 3-UPLES ((2 2 2) (1 2 1))
 OK
```

Et voici le listing de MINSER :

```
(DM SETA (call)
   (RPLACB call ['SET ['NTH (CADDR call) (CADR call)]
                   (CADR (CDDR call))]))

(DM ELM (call) (RPLACB  call
   ['CAR ['NTH (CADDR call) (CADR call)]]))

(DM SEND (call) (RPLACB call
   ['SETQ '#target (CADR call) '#msg (CADDR call)]))

(DE M-SERIES (let)
   (SETQ ls 0 lk 0 status () start 0 3-uples NIL)
   (SEND '$initialiser let)
   (WHILE (NEQ #target '$fini)  ; loop loop loop ;
      (#target #msg))
    (PRINT #msg let)
    (PRINT '3-uples 3-uples)
    'OK)

(DE $initialiser (let)
   (IF (NULL let) (SEND '$newstart (ADD1 start))
        (SETQ lk (ADD1 lk) status (CONS NIL status))
        (IF (NEQ (NEXTL let) '?) (SETQ ls (ADD1 ls)))
        ($initialiser let)))
```

```
(DE $newstart (msg)
   (SETQ start msg)
   (COND
        ((GT start lk) (SEND '$fini 'succes))
        ((ELM status start) ($newstart (ADD1 start)))
        (T (SEND '$newmove 1))))

(DE $newmove (msg)
   (SETQ move msg)
   (IF (GT (PLUS start move move) ls) (SEND '$fini 'echec)
        (SETQ inc (DIFFER (SETQ predicted (ELM let (PLUS start move)))
                          (ELM let start))
              predicted (PLUS predicted inc))
        (SEND '$verifier (PLUS start move move))))

(DE $verifier (x) (COND
   ((GT x LS)
        (SETQ 3-uples (CONS [start move inc] 3-uples))
        (SEND '$completer start))
   ((NEQ predicted (ELM let x)) (SEND '$newmove (ADD1 move)))
   (T (SETQ predicted (PLUS predicted inc))
        ($verifier (PLUS x move)))))

(DE $completer (x)
   (IF (GT x lk) (SEND '$newstart (ADD1 start))
        (SETA  status  x T)
        (AND (GT x ls)
           (SETA let x predicted)
           (SETQ predicted (PLUS predicted inc)))
        ($completer  (PLUS x move))))

(PRINT 'Lancer 'avec '(M-SERIES '(x x x x ? ? ?)))
```

9.2 UN INTERPRETE VLISP PUR ET SON OPERATING SYSTEM

Les seules fonctions standard connues de cet interprete sont DE, CAR, CDR, CONS, ATOM, EQ, QUOTE et IF.

La technique des A-listes est ici utilisee pour implementer les environnements, et realise le type de liaison dit PROFOND.

L'Operating System est la fonction LOOP chargee de lire les definitions (elle-seule peut comprendre les appels de DE), de passer a SEVAL les autres expressions lues (atomes ou appels), d'imprimer enfin le resultat de l'evaluation.

L'appel de LOOP dans le corps de LOOP est un cas de fausse recursivite, interprete sans consommation de pile, tels que ceux decrits au Chapitre 1, Paragraphe 4.

Que le lecteur se persuade bien que VLISP 16 n'est PAS implemente de cette maniere, et c'est heureux.

```
(DE VALUE (V ENV)                   ; Pour extraire de l'ENVironnement ;
  (IF (EQ V (CAAR ENV))             ; la valeur d'une VARiable.        ;
      (CADAR ENV)
      (VALUE V (CDR ENV))))


(DE NEWENV (V E ENV)                ; Pour ajouter a l'ENVironnement   ;
  (CONS [V E] ENV))                 ; un nouveau couple (<var> <e>)    ;


(DE SUPNEWENV (X Y ENV)             ; Avec X = (<v1> ... <vN>)         ;
  (IF (NULL X) ENV                  ; et   Y = (<e1> ... <eN>).        ;
      (NEWENV (NEXTL X) (NEXTL Y) (SUPNEWENV X Y ENV))))


(DE LISP ()                         ; Lance tout.                      ;
  (LOOP 'DEBUT (SREAD) '((NIL NIL) (T T))))


(DE LOOP (SORTIE ENTREE ENV)        ; L'Operating System               ;
  (IF (AND (LISTP ENTREE) (EQ (CAR ENTREE) 'DE))
      (LOOP (PRINT (CADR ENTREE))
            (SREAD)
            (NEWENV (CADR ENTREE)
                    (CONS LAMBDA (CDDR ENTREE))
                    ENV))
      (LOOP (PRINT (SEVAL ENTREE ENV)) (SREAD) ENV)))

(DE SREAD ()                        ; Pour lire en etant prevenu.      ;
  (PRINT 'QUELQUE 'CHOSE 'A 'EVALUER 'SVP)
  (READ))
```

```
(DE ST-SUBR (F)                     ; F est-elle une SUBR ?            ;
  (MEMQ F '(CAR CDR CONS ATOM EQ)))

(DE XCT-ST-SUBR (F L)               ; Appliquez la SUBR F a la liste   ;
  (SELECTQ F                        ; L des arguments evalues          ;
    (CAR  (CAAR L))
    (CDR  (CDAR L))
    (ATOM (ATOM (CAR L)))
    (CONS (CONS (CAR L) (CADR L)))
    (EQ   (EQ (CAR L) (CADR L)))
    ()))


(DE ST-FSUBR (F)                    ; F est-elle une FSUBR ?           ;
  (MEMQ F '(IF QUOTE)))


(DE XCT-ST-FSUBR (E ENV)            ; Pour evaluer l'appel E de FSUBR  ;
  (SELECTQ (CAR E)
    (QUOTE (CADR E))
    (IF    (IF (SEVAL (CADR E) ENV)
               (SEVAL (CADDR E) ENV)
               (SEVAL (CADR (CDDR E)) ENV)))
    ()))


(DE SEVAL (E ENV) (COND             ; Le EVAL de ce LISP               ;
  ((ATOM E) (VALUE E ENV))
  ((ATOM (CAR E))
     (IF (ST-FSUBR (CAR E))
         (XCT-ST-FSUBR E ENV)
         (SAPPLY (NEXTL E) (EVARGS E ENV) ENV)))
  (T (SAPPLY (NEXTL E) (EVARGS E ENV) ENV))))


(DE SAPPLY (F ARGS ENV) (COND       ; Le APPLY de ce LISP              ;
  ((ATOM F) (IF (ST-SUBR F)
                (XCT-ST-SUBR F ARGS)
                (SAPPLY (SEVAL F ENV) ARGS ENV)))
  ((EQ (CAR F) LAMBDA)
   (SEVAL (CADDR F) (SUPNEWENV (CADR F) ARGS ENV)))
        ; pour avoir SELF : remplacer le ENV precedent par : ;
        ; (NEWENV 'SELF F ENV)                               ;
  (T (SAPPLY (SEVAL F ENV) ARGS ENV))))


(DE EVARGS (LARGS ENV)              ; Pour construire la liste    ;
  (IF (NULL LARGS) NIL              ; des elements de LARGS evalues ;
      (CONS (SEVAL (NEXTL LARGS) ENV)
            (EVARGS LARGS ENV))))

(PRINT 'Pour 'lancer 'faire '(LISP))
```

9.3 UNE MACHINE LISP VIRTUELLE

Il s'agit d'une simplification considerable de la machine LISP de Jerome CHAILLOUX nommee VCMC2.

Petite machine-LISP donc, comportant les registres A1, A2, A3, A4, A5, ENV. Avec de surcroit un compteur ordinal PC et une pile cablee dont nous appellerons P le registre pointeur de sommet.

En voici les instructions :

```
(MV <e> <reg>)         <e> → <reg>

    avec <reg> ::= A1|A2|A3|A4|A5|ENV
    et   <e>   ::= <reg>
                  |T
                  |NIL
                  |'<une-expression>
                  |(CAR <e>)
                  |(CDR <e>)
                  |(<n> P)  ~adressage relatif au
                            ~sommet de pile.

(PUSH <e>)             incr P; <e> → M[P]

(POP <reg>)            M[P] → <reg>; decr P

(SUB P <n>)            P - <n> → P

(REC <etiq>)           incr P; PC → M[P]; <etiq> → PC

(DEREC)                M[P] → PC; decr P

(JATOM <e> <etiq>)     <etiq> → PC si (ATOM <e>)

(JNATOM <e> <etiq>)    <etiq> → PC si (NOT (ATOM <e>))

(JEQ <e1> <e2> <etiq>)
                       <etiq> → PC si <e1> = <e2>

(JNEQ <e1> <e2> <etiq>)
                       <etiq> → PC si <e1> ≠ <e2>

(J <etiq>)             <etiq> → PC

(CONS <e1> <e2> <reg>)
                       <e1> CONS <e2> → <reg>

(PRINT <e>)            impression de <e>

(READ <reg>)           lire dans <reg>
```

Pour lancer toute l'affaire faites :

```
(RUN 'LISP)
```

Et voila le listing de la micro-machine.

```
(DE EXP (X) (COND                 ; Pour decoder une <e>            ;
     ((ATOM X) (COND               ; sans trop chercher a verifier ;
                  ((MEMQ X '(T NIL)) X)
                  ((ISREG X) (CAR X))
                  (T (ERREUR 'REG))))
     ((NUMBP (CAR X))
      (LET ((N (CAR X)) (STACK STACK))
        (IF (ZEROP N) (CAR STACK)
            (SELF (1+ N) (CDR STACK)))))
     ((EQ (CAR X) 'QUOTE) (CADR X))
     ((MEMQ (CAR X) '(CAR CDR CAAR CADR CDAR CDDR))
      (EVAL X))
     (T (ERREUR 'EXP))))


(DE ERREUR (K) (PRINT 'ERREUR K)
               (WHILE T (PRINT (EVAL (READ)))))

(DE ISREG (X) (MEMQ X '(A1 A2 A3 A4 A5 ENV)))

(DE PUSH (X) (SETQ STACK (CONS X STACK)))

(DE POP () (NEXTL STACK))

(DE RUN (START) ; START : etiquette de depart dans MEM ;
  (SETQ A1 (SETQ A2 (SETQ A3 (SETQ A4 (SETQ A5 'START)))))
  (SETQ STACK NIL)
  (SETQ PC (MEMQ START MEM))
  (LOOP))

(SETQ STEP NIL) ; Pour Stepper chanter (SETQ STEP T)     ;


(DE LOOP ()      ; La boucle de decodage et de lancement ;
   (WHILE PC
         (IF (ATOM (CAR PC)) (NEXTL PC)
             (XCT (NEXTL PC)))))

(DE JUMP (X ;; Y) (SETQ Y (MEMQ X MEM))
                  (IF Y (SETQ PC Y) (ERREUR 'ETIQ-INC)))
```

```
(DE XCT (INS)    ; Les micro-programmes des instructions ;
  (SELECTQ (CAR INS)
    (PUSH (PUSH (EXP (CADR INS))))
    (POP  (SET (CADR INS) (POP)))
    (SUB  (SETQ STACK (NTH (ADD1 (CADDR INS)) STACK)))
    (REC  (PUSH PC) (JUMP (CADR INS)))
    (MV   (SET (CADDR INS) (EXP (CADR INS))))
    (DEREC (SETQ PC (POP)))
    (JATOM (IF (ATOM (EXP (CADR INS))) (JUMP (CADDR INS))))
    (JNATOM (IF (ATOM (EXP (CADR INS))) NIL (JUMP (CADDR INS))))
    (JNEQ   (IF (NEQ (EXP (CADR INS)) (EXP (CADDR INS)))
                (JUMP (CADR (CDDR INS)))))
    (JEQ    (IF (EQ (EXP (CADR INS)) (EXP (CADDR INS)))
                (JUMP (CADR (CDDR INS)))))
    (J      (JUMP (CADR INS)))
    (CONS   (SET (CADR (CADDR INS))
                 (CONS (EXP (CADR INS)) (EXP (CADDR INS)))))
    (PRINT  (PRINT (EXP (CADR INS))))
    (READ   (PRINT 'SMTHNG 'TO 'EAT 'PLEASE '?)
            (SET (CADR INS) (READ)))
    (T (ERREUR 'INSTR-INC))))
```

Et voici a present l'interprete VLISP pur et son operating system code dans le langage assembleur LAP specifique a cette machine.

```
(SETQ MEM '(

LISP
         (MV '((NIL NIL) (T T)) ENV)      ; envir. initial ;
         (J LOOPTEST)                     ; en voiture !   ;
LOOP1                            ; l'expression lue n'est   ;
                                 ; pas une definition.      ;
         (REC EVAL)              ; evaluons-la donc         ;
LOOP2
         (PRINT A1)              ; imprimer le resultat  ;
LOOPTEST                         ; et recommencer        ;
         (READ A1)               ; lire une expression   ;
         (JATOM A1 LOOP1)        ; est-ce un atome ?     ;
         (JNEQ (CAR A1) 'DE LOOP1);est-ce une definition ? ;
         (MV (CDDR A1) A2)       ; c'est une (DE nom largs corps)  ;
         (CONS 'LAMBDA A2 A2)    ; A2 = (LAMBDA largs corps)       ;
         (MV (CADR A1) A1)       ; A1 = le-nom           ;
         (PUSH A1)
         (MV ENV A3)             ; preparer A3 pour NEWENV ;
         (REC NEWENV)
         (MV A1 ENV)             ; ENV = ((nom lambda) . old-ENV) ;
         (POP A1)                ; A1  = le-nom pour impression   ;
         (J LOOP2)
NEWENV                           ; A1 = un-nom . A2 = une-valeur  ;
                                 ; A3 = un-environnement.        ;
         (CONS A2 NIL A2)
         (CONS A1 A2 A1)
         (CONS A1 A3 A1)
         (DEREC)                 ; A1 = ((nom valeur) . env)      ;

VALUE                            ; A1 = un-nom. On va chercher    ;
                                 ; sa valeur dans ENV             ;
         (MV ENV A2)             ; preserver ENV dans A2          ;
         (J VALUETEST)           ; et commencer la recherche      ;
VALUE2
         (MV (CDR A2) A2)        ; avancer dans l'environnement   ;
VALUETEST
         (JEQ A2 NIL ERREUR)     ; on n'a pas trouve le nom !!!   ;
         (JNEQ (CAAR A2) A1 VALUE2) ; est-ce le nom ?             ;
         (MV (CAR(CDAR A2)) A1)  ; oui. A1 = la-valeur associee   ;
         (DEREC)

SUPNEWENV                        ; A1 = une liste de noms nN      ;
                                 ; A2 = une liste de valeurs vN   ;
                                 ; A3 = un environnement e        ;
         (JNEQ A1 NIL SUPNEW2)
```

```
             (MV A3 A1)
             (DEREC)                   ; A1 = ((n1 v1) ... (nN vN) . e) ;
SUPNEW2
             (PUSH (CAR A1))
             (PUSH (CAR A2))
             (MV (CDR A1) A1)
             (MV (CDR A2) A2)
             (REC SUPNEWENV)
             (MV A1 A3)
             (POP A2)
             (POP A1)
             (J NEWENV)

ST-SUBR                                ; A1 est-il CAR ou CDR ou CONS   ;
                                       ; ou ATOM ou EQ ?                ;
             (JEQ A1 'CAR TRUE)
             (JEQ A1 'CDR TRUE)
             (JEQ A1 'CONS TRUE)
             (JEQ A1 'ATOM TRUE)
             (JEQ A1 'EQ TRUE)
FALSE
             (MV NIL A1)
             (DEREC)
TRUE
             (MV T A1)
             (DEREC)
EXEC-ST-SUBR                           ; aiguillage selon la SUBR       ;
                                       ; A1 = la-SUBR                   ;
                                       ; A2 = une liste d'arguments     ;
                                       ;      evalues.                  ;
             (JEQ A1 'CAR CAR)
             (JEQ A1 'CDR CDR)
             (JEQ A1 'CONS CONS)
             (JEQ A1 'ATOM ATOM)
             (JEQ A1 'EQ EQ)
             (J ERREUR)                ; SUBR inconnue !!!              ;
CAR
             (MV (CAAR A2) A1)
             (DEREC)
CDR
             (MV (CDAR A2) A1)
             (DEREC)
CONS
             (CONS (CAR A2) (CADR A2) A1)
             (DEREC)
ATOM
             (JATOM (CAR A2) TRUE)
             (J FALSE)
EQ
             (JEQ (CAR A2) (CADR A2) TRUE)
             (J FALSE)
```

```
ST-FSUBR                              ; A1 est-il QUOTE ou IF ?       ;
          (JEQ A1 'QUOTE TRUE)
          (JEQ A1 'IF TRUE)
          (J FALSE)

EXEC-ST-FSUBR                         ; A1 = (QUOTE e) ou             ;
                                      ; A1 = (IF test e1 e2) ou       ;
                                      ; A1 = ???                      ;
          (JEQ (CAR A1) 'QUOTE QUOTE)
          (JEQ (CAR A1) 'IF IF)
          (J ERREUR)
QUOTE
          (MV (CADR A1) A1)           ; A1 = e                        ;
          (DEREC)
IF
          (PUSH (CDDR A1))            ; preserver (e1 e2)             ;
          (MV (CADR A1) A1)           ; A1 = le-test                  ;
          (REC EVAL)                  ; l'evaluer                     ;
          (POP A2)                    ; A2 = (e1 e2)                  ;
          (JEQ A1 NIL IF2)
          (MV (CAR A2) A1)            ; si test OK evaluer e1         ;
          (J EVAL)
IF2
          (MV (CADR A2) A1)           ; si test KO evaluer e2         ;
          (J EVAL)


EVAL                                  ; l'evaluateur en personne      ;
                                      ; A1 = une expression e         ;
          (JATOM A1 VALUE)            ; si c'est un nom ramener       ;
                                      ; sa valeur dans ENV.           ;

          (PUSH A1)                   ; e = (fonc a1 a2 ... aN)       ;
          (JNATOM (CAR A1) EVAL2)     ; e est il atomique ?           ;
          (MV (CAR A1) A1)            ; oui. A1 = la-fonction         ;
          (REC ST-FSUBR)              ; est-ce une FSUBR ?            ;
          (JEQ A1 NIL EVAL2)
          (POP A1)                    ; oui. A1 = (la-FSUBR a1 ... aN) ;
          (J EXEC-ST-FSUBR)           ; l'executer froidement.        ;

EVAL2                                 ; la-fonction n'est pas une FSUBR ;
                                      ; on va evaluer les arguments   ;
                                      ; (a1 ... aN) . Ca sera toujours ;
                                      ; ca de fait.                   ;

          (POP A1)                    ; recuperer A1 = (fonc a1 ... aN) ;
          (PUSH (CAR A1))             ; preserver la-fonction.        ;
          (MV (CDR A1) A1)            ; A1 = (a1 ... aN)              ;
          (REC EVARGS)
          (MV A1 A2)                  ; A2 = (val-a1 ... val-aN)      ;
          (POP A1)                    ; A1 = la-fonction.             ;
          (J APPLY)                   ; aux bons soins d'APPLY.       ;
```

```
EVARGS                                  ; a l'entree, A1 = (a1 ... aN)        ;
          (JEQ A1 NIL FALSE)
          (PUSH A1)
          (MV (CAR A1) A1)
          (REC EVAL)
          (POP A2)
          (PUSH A1)
          (MV (CDR A2) A1)
          (REC EVARGS)
          (POP A2)
          (CONS A2 A1 A1)
          (DEREC)                       ; a la sortie,                        ;
                                        ; A1 = (val-a1 ... val-aN)            ;

APPLY                                   ; A1 = une-supposee-fonction.         ;
                                        ; A2 = une liste d'arguments          ;
                                        ;      evalues.                       ;

          (PUSH A1)                     ; preserver la supposee-fonction      ;
          (JNATOM A1 APPLY2)            ; la fonction est-elle atomique ?     ;
          (REC ST-SUBR)                 ; oui. Est-ce une SUBR ?              ;
          (JEQ A1 NIL APPLY3)
          (POP A1)                      ; oui. L'appliquer froidement         ;
          (J EXEC-ST-SUBR)              ; a la liste de valeurs d'args.       ;

APPLY2                                  ; le fonc. n'est pas une SUBR.        ;
          (JNEQ (CAR A1) 'LAMBDA APPLY3)
                                        ; est-ce une LAMBDA-expression ?      ;
                                        ; i.e. (LAMBDA (v1 ... vN) corps)     ;
          (MV (CADR A1) A1)             ; oui. A1 = (v1 ... vN)               ;
          (MV ENV A3)
          (REC SUPNEWENV)
          (POP A2)
          (PUSH ENV)                    ; sauver l'environnement courant      ;
          (MV A1 ENV)                   ; Voici le nouvel environnement :     ;
                                        ; ((v1 val-a1) ...                    ;
                                        ;  (vN val-aN) . le-vieil-ENV)        ;
          (MV A2 A1)
          (MV (CAR (CDDR A1)) A1)       ; A1 = le-corps de la LAMBDA-exp.     ;
          (REC EVAL)                    ; qu'on evalue dans le                ;
                                        ; nouvel environnement.               ;

          (POP ENV)                     ; A1 = le resultat de                 ;
                                        ;      l'evaluation du corps.         ;
                                        ; On restitue l'ancien ENV.           ;
          (DEREC)                       ; Et on retourne a l'envoyeur.        ;

APPLY3                                  ; le-fonction est donc un quasar      ;
                                        ; qu'il convient d'evaluer.           ;
          (POP A1)                      ; A1 = le-fonction.                   ;
          (PUSH A2)                     ; sauvons la liste des                ;
                                        ; (val-a1 ... val-aN)                 ;
          (REC EVAL)                    ; evaluons le quasar.                 ;
```

```
        (POP A2)            ; restituons la liste des val-al    ;
        (J APPLY)           ; et reprenons l'examen             ;
                            ; de la nature de la-fonction.      ;
                            ; DE FONCTIONA RERUM                ;

  ))
```

9.4 LISTING DE L'EDITEUR EF, DE ADVISE ET BREAK.

Ces outils de mise au point ont eta largement exposes au Chapitre 8. Le lecteur s'y reportera en cas de doute.

```
(DF MV (L ;; X)
  (WHILE L (SETQ X (NEXTL L))
        (SETQ ** (IF (NUMBP X) (NTH X **) (CAR **))))
  (P 1))

(DE SPRINT (N L)        ; Pour imprimer l'expression L sur N niveaux ;
  (PRINT (IF (OR (ATOM L) (ZEROP N)) L (SPRC L 0 N))))

(DE SPRC (L N1 N2) (COND
  ((NULL L) NIL)
  ((GT N1 N2) '*)
  ((ATOM L) L)
  ((CONS (SPRC (NEXTL L) (ADD1 N1) N2) (SPRC L N1 N2)))))

(DE FIND (L A P) (COND          ; Pour aider FK et FP ;
  ((ATOM L) NIL)
  ((AND A (EQUAL (CAR L) A)) L)
  ((AND P (FILTER (CAR L) P)) L)
  ((OR (FIND (NEXTL L) A P) (FIND L A P)))))

(DE FILTER (D P) (COND
  ((EQ P '?))            ; Ex. d'un tel filtre : (COND . ?) ;
  ((NULL P) (NULL D))
  ((ATOM (CAR P))
   (IF (OR (EQ (CAR P) '?) (EQ (CAR P) (CAR D)))
       (FILTER (CDR D) (CDR P))))
  ((FILTER (NEXTL D) (NEXTL P)) (FILTER D P))))

(DF EF (F) (SETQ ** (CADDR (CAR F))) (P 1))     ; L'editeur lui-meme ;

(DE P (N) (SPRINT N **) '*)

(DF IL (E)         ; (IL e1 e2 ... eN) ;
  (NCONC ** E)
  (P 1))

(DF I (L)
  (NCONC L (CONS (CAR **) (CDR **)))
  (RPLACB ** L)
  (P 1))

(DF FP (F)
  (SETQ ** (FIND ** NIL (CAR F)))
  (P 1))
```

```
(DF FK (A)
  (SETQ ** (FIND ** (CAR A) NIL))
  (P 1))

(DE D (N) (REPEAT N (RPLACB ** (CDR **)))
  (P 1))


(DF DL ( ;; X)
  (SETQ X **) (WHILE (CDDR X) (NEXTL X)) (RPLACD X) (P 1))

(DF ADVISE (L ;; NOM EXP RES1)
  (SETQ NOM (NEXTL L) EXP (CADDR NOM) RES1 ['PROGN])
  (PUT NOM EXP 'ADVISE)
  (WHILE (NEQ (CAR L) '*) (NCONC1 RES1 (NEXTL L)))
  (NEXTL L)
  (RPLACA (CDDR NOM)
          (MCONS LAMBDA (CADR EXP)
                 ['SETQ '-VAL- (NCONC RES1 (CDDR EXP))]
                 (NCONC1 L '-VAL-)))
  (CONS NOM))


(DF UNADVISE (L)
  (RPLACA (CDDR (CAR L)) (GET (CAR L) 'ADVISE))
  (REMPROP (CAR L) 'ADVISE)
 L)

(DF BREAK (L ;; NOM X)
  (SETQ NOM (CAR L) X (CADDR NOM))  ; X = la λ-expression ;
  (PUT NOM X 'BREAK)
  (SET (CDDR NOM) (NCONC [LAMBDA (CADR X)
                                 ['TOP [QUOTE ['BREAK NOM]]]]
                         (CDDR X)))
 L)

(DF UNBREAK (L)
  (SET (CDDR (CAR L)) (GET (CAR L) 'BREAK))
  (REMPROP (CAR L) 'BREAK)
 L)

(DE TOP (MSG ;; -X-)             ; La boucle TOP-LEVEL du BREAK ;
  (PRINT MSG)
  (IF (EQ (SETQ -X- (READ)) T) T
      (PRINT (EVAL -X-))
      (TOP MSG)))
```

9.5 UN PROGRAMME DE VERIFICATION DE FORMULES EN CALCUL PROPOSITIONNEL

Ce programme est une adaptation en VLISP 16 de la version livree dans le Cours d'Option au fascicule 5 pp. 54-55. La verification des formules est effectuee par la methode dite "des tableaux".

Et voici quelques verifications :

```
?(THEOREME '(A OU (PAS A)))
 T

?(THEOREME '(A IMP (B IMP A)))
 T

?(THEOREME '((A IMP B) IMP ((PAS B) IMP (PAS A))))
 T

?(THEOREME '(((A IMP B) ET A) IMP B))
 T

?(THEOREME '(((A OU B) ET (PAS A)) IMP B))
 T

?(THEOREME '(((PAS A) OU (B IMP C)) IMP ((A ET (PAS B)) OU (A IMP C))))
 T
```

Le listing du verifieur suit :

```
(DE THEOREME (L)
   (FERME () [(CONS '- (POL L))]))

(DE POL (L) (COND         ; Pour rendre les formules illisibles ;
   ((ATOM L) [L])
   ((EQ (CAR L) 'PAS) L)
   (T (CONS (CADR L) (APPEND (POL (CAR L)) (POL (CADDR L)))))))

(DE FERME (X L) (COND
   ((NULL L) NIL)
   ((MEMQ (CADAR L) '(PAS IMP ET OU))
    ((CADAR L) X (CAAR L) (CDDAR L) (CDR L)))
   (T (AJOUT X [(CAR L)] (CDR L)))))
```

```
(DE AJOUT (X L1 L2) (COND
  ((NULL L1) (FERME X L2))
  ((MEMQ (CADAR L1) '(PAS IMP ET OU))
   (COND ((MEMBER (CAR L1) L2) (AJOUT X (CDR L1) L2))
         ((MEMBER (CONS (NEGAT (CAAR L1)) (CDAR L1)) L2) T)
         (T (AJOUT X (CDR L1) (CONS (CAR L1) L2)))))
  ((MEMBER (CAR L1) X) (AJOUT X (CDR L1) L2))
  ((MEMBER (CONS (NEGAT (CAAR L1)) (CDAR L1)) X) T)
  (T (AJOUT (CONS (NEXTL L1) X) L1 L2))))

(DE NEGAT (V) (IF (EQ V '+) '- '+))

(DE SOUSEXP (L1 L2 N)
  (IF (ZEROP N) (CONS L1 L2)
      (SOUSEXP (APPEND L1 [(CAR L2)])
               (CDR L2)
               (COND  ((EQ (CAR L2) 'PAS) N)
                      ((MEMQ (CAR L2) '(ET OU IMP)) (ADD1 N))
                      (T (SUB1 N))))))

(DE PAS (X V L1 L2)
  (AJOUT X [(CONS (NEGAT V) L1)] L2))

(DE ET (X V L1 L2)
  (LET ((Y (SOUSEXP NIL L1 1)))
       (IF (EQ V '+)
           (AJOUT X [(CONS V (CAR Y)) (CONS V (CDR Y))] L2)
           (AND (AJOUT X [(CONS V (CAR Y))] L2)
                (AJOUT X [(CONS V (CDR Y))] L2)))))

(DE OU (X V L1 L2)
  (LET ((Y (SOUSEXP NIL L1 1)))
       (IF (EQ V '+)
           (AND (AJOUT X [(CONS V (CAR Y))] L2)
                (AJOUT X [(CONS V (CDR Y))] L2))
           (AJOUT X [(CONS V (CAR Y)) (CONS V (CDR Y))] L2))))

(DE IMP (X V L1 L2)
  (LET ((Y (SOUSEXP NIL L1 1)))
       (IF (EQ V '+)
           (AND (AJOUT X [(CONS (NEGAT V) (CAR Y))] L2)
                (AJOUT X [(CONS V (CDR Y))] L2))
           (AJOUT X [(CONS (NEGAT V) (CAR Y)) (CONS V (CDR Y))] L2))))
```

9.6 UN PROGRAMME DE DEMONSTRATION INTERACTIVE.

Ce programme permet de demontrer interactivement des proprietes de fonctions LISP.
Ya tout le tremblement : evaluation symbolique, induction structurelle, generalisation, subst. de l'hypothese d'induction ...

Les commandes:

-1- definitions de fonctions VLISP ordinaires (peuvent s'inter-appeller).

-2- (ANA <le-nom-de-la-fonction>), quoter le nom SVP.
Suit obligatoirement la definition, si la fonction est utilisee dans un theoreme (ca ne l'empeche pas de tourner).

-3- (THEO <partie-gauche> = <partie-droite>)
pour soumettre le theoreme a l'interactiveteoremprouvere.

-4- (INDUCT LIST <nom-de-variable>) , induction type-liste.

-5- (INDUCT NUM <nom-de-variable>) , itou sur type-nombre.

-6- (GEN <expressionI> <varI> ...) pour i = 1,n.
Substitue (generalisation) les expressionsI aux variablesI.

-7- (INDHYP <n> <direction>)
Substitue a la n-ieme occurence de la partie gauche de l'hypothese d'induction (direction = "->") sa partie droite.
L'inverse si direction = "<-" .

Exemple de session:

```
        ; On definit une fonction APP ;
(DE APP (X Y) (IF (NULL X) Y (CONS (CAR X) (APP (CDR X) Y))))
        ; On en demande une analyse succinte ;
(ANA 'APP)
        ; On rentre le theoreme qu'on souhaite demontrer ;
(THEO (APP X (APP Y Z)) = (APP (APP X Y) Z))
        ; Faut faire une induction structurelle de type-liste
          sur la variable X ;
(INDUCT LIST X)
        ; Faut utiliser l'hypothese d'induction en
          substituant sa partie droite (->) a sa partie gauche,
          et sur la 1ere occurence de la partie gauche;
(INDHYP 1 ->)
        ; Voila le theoreme demontre !!! ;
```

Que les primitives recursives soient avec vous.

Voici a present deux exemples d'utilisation du programme.

```
?(THEO (APP A (APP B C)) = (APP (APP A B) C))
 ** 0 PROVE (APP A (APP B C)) = (APP (APP A B) C)
 COMMAND ?
?(INDUCT LIST A)
 ** BASE
 ** 1 PROVE (APP NIL (APP B C)) = (APP (APP NIL B) C)
 ** 1 PROVED
 ** STEP
 ** 1 PROVE (APP (CONS G101 A) (APP B C)) = (APP (APP (CONS G101 A) B) C)
 ** 2 PROVE (CONS G101 (APP A (APP B C))) = (CONS G101 (APP (APP A B) C))
 COMMAND ?
?(INDHYP 1 ->)
 ** 3 PROVE (CONS G101 (APP (APP A B) C)) = (CONS G101 (APP (APP A B) C))
 ** 3 PROVED
 ** 2 PROVED
 ** 1 PROVED
 ** 0 PROVED
 PROVED
```

```
?(THEO (P A B) = (P B A))
 ** 0 PROVE (P A B) = (P B A)
 COMMAND ?
?(INDUCT NUM A)
 ** BASE
 ** 1 PROVE (P 0 B) = (P B 0)
 ** 2 PROVE B = (P B 0)
 COMMAND ?
?(INDUCT NUM B)
 ** BASE
 ** 3 PROVE 0 = (P 0 0)
 ** 3 PROVED
 ** STEP
 ** 3 PROVE (ADD1 B) = (P (ADD1 B) 0)
 ** 4 PROVE (ADD1 B) = (ADD1 (P B 0))
 COMMAND ?
?(INDHYP 1 ->)
 ** 5 PROVE (ADD1 (P B 0)) = (ADD1 (P B 0))
 ** 5 PROVED
 ** 4 PROVED
 ** 3 PROVED
 ** 2 PROVED
 ** 1 PROVED
 ** STEP
 ** 1 PROVE (P (ADD1 A) B) = (P B (ADD1 A))
 ** 2 PROVE (ADD1 (P A B)) = (P B (ADD1 A))
 COMMAND ?
?(INDHYP 1 ->)
 ** 3 PROVE (ADD1 (P B A)) = (P B (ADD1 A))
 COMMAND ?
?(INDUCT NUM B)
 ** BASE
 ** 4 PROVE (ADD1 (P 0 A)) = (P 0 (ADD1 A))
 ** 4 PROVED
 ** STEP
 ** 4 PROVE (ADD1 (P (ADD1 B) A)) = (P (ADD1 B) (ADD1 A))
 ** 5 PROVE (ADD1 (ADD1 (P B A))) = (ADD1 (P B (ADD1 A)))
 COMMAND ?
?(INDHYP 1 ->)
 ** 6 PROVE (ADD1 (P B (ADD1 A))) = (ADD1 (P B (ADD1 A)))
 ** 6 PROVED
 ** 5 PROVED
 ** 4 PROVED
 ** 3 PROVED
 ** 2 PROVED
 ** 1 PROVED
 ** 0 PROVED
 PROVED
```

Voici le listing du demonstrateur interactif.

```
(DE ANA (F ;; X VARS BODY DECVARS)
  (SETQ X (GET F 'EXPR)
        VARS (CADR X)
        BODY (CADDR X))
  (LET ((E BODY)) (COND
    ((ATOM E))
    ((MEMQ (CAR E) '(CAR CDR SUB1))
     (IF (ATOM (CADR E))
         (AND (MEMQ (CADR E) VARS)
              (OR (MEMQ (CADR E) DECVARS)
                  (SETQ DECVARS (CONS (CADR E) DECVARS))))
         (SELF (CADR E))))
    (T (SELF (NEXTL E)) (SELF E))))
  (PUT F VARS 'VARS)
  (PUT F BODY 'BODY)
  (PUT F DECVARS 'DECVARS))

(DE VARS (F) (GET F 'VARS))
(DE BODY (F) (GET F 'BODY))
(DE DECVARS (F) (GET F 'DECVARS))
```

```
; ****** EVALUATEUR SYMBOLIQUE ******* ;

(DE RED (E AL) (COND
  ((ATOM E) (LET ((VAL (ASSQ E AL)))
                (IF VAL (CADR VAL) E)))
  ((EQ (CAR E) 'IF)
   (LET ((TEST (RED (CADR E) AL))) (COND
      ((NULL TEST) (RED (CADR(CDDR E)) AL))
      ((OR (EQ TEST T) (EQ (CAR TEST) 'CONS))
       (RED (CADDR E) AL))
      (T (MCONS 'IF TEST (CDDR E))))))
   (T (LET ((LARGS (MAPCAR (CDR E) (LAMBDA (E) (RED E AL))))
            (FUN (CAR E)))
        (SELECTQ FUN
          (CAR (IF (EQ (CAAR LARGS) 'CONS)
                   (CADAR LARGS)
                   (CONS FUN LARGS)))
          (CDR (IF (EQ (CAAR LARGS) 'CONS)
                   (CADDR (CAR LARGS))
                   (CONS FUN LARGS)))
          (SUB1 (IF (EQ (CAAR LARGS) 'ADD1)
                    (CADAR LARGS)
                    (CONS FUN LARGS)))
          (ADD1 (CONS FUN LARGS))
          (CONS (CONS FUN LARGS))
          (ZEROP (COND ((ZEROP (CAR LARGS)) T)
                       ((EQ (CAAR LARGS) 'ADD1) NIL)
                       (T (CONS FUN LARGS))))
          (NULL  (COND ((NULL (CAR LARGS)) T)
                       ((EQ (CAR LARGS) T) NIL)
                       ((EQ (CAAR LARGS) 'CONS) NIL)
                       (T (CONS FUN LARGS))))
          (T (LET ((AL1 (REDPAIR FUN (VARS FUN) LARGS)))
                 (IF (EQ AL1 'NO) (CONS FUN LARGS)
                     (RED (BODY FUN) (NCONC AL1 AL))))))))))

; ****** CONTROLEUR DE DEPLOIEMENT ****** ;

(DE REDPAIR (FUN VARS LARGS ;; DECVARS RES V L)
  (ESCAPE EXIT
    (SETQ DECVARS (DECVARS FUN))
    (WHILE VARS
      (SETQ V (NEXTL VARS) L (NEXTL LARGS))
      (IF (MEMQ V DECVARS)
          (AND L (NEQ L 0)
               (NEQ (CAR L) 'CONS) (NEQ (CAR L) 'ADD1)
               (EXIT 'NO)))
       (SETQ RES (CONS (LIST V L) RES)))
    RES))
```

```
; ********* QUELQUES FONCTIONS DE TEST ********* ;

(DE LOOP () (PRINT 'EVAL 'SYM ) (PRINT (RED (READ) ()))) (TERPRI) (LOOP))

(DE APP (X Y) (IF (NULL X) Y (CONS (CAR X) (APP (CDR X) Y))))
(ANA 'APP)

(DE REV (X) (IF (NULL X) NIL (APP (REV (CDR X)) (CONS (CAR X) NIL))))
(ANA 'REV)

(DE LEN (L) (IF (NULL L) 0 (ADD1 (LEN (CDR L)))))
(ANA 'LEN)

(DE P (X Y) (IF (ZEROP X) Y (ADD1 (P (SUB1 X) Y))))
(ANA 'P)

(DE M (X Y) (IF (ZEROP X) 0 (P Y (M (SUB1 X) Y))))
(ANA 'M)
```

```
; ******** LE-PROOF-VERIFIEUR ******** ;

(DF THEO (L)
  (PROVE L 0))


; EX: (THEO (P M N) = (P N M)) ;

(DE PROVE (TH N ;; THH)  ; TH = ( * = ** ) ;
  (PSTAR (NCONC [N 'PROVE] TH))
  (COND
    ((EQ (SETQ THH (REDUCE TH)) T))
    ((EQUAL TH THH) (COMMAND))
    (T (PROVE THH (ADD1 N))))
  (PSTAR [N 'PROVED]))

(DE PSTAR (L)
  (PRIN1 '**)
  (APPLY 'PRINT L))


(DE COMMAND (;; COM PAT REMP NBOC)
  (PRINT 'COMMAND '/?)
  (SETQ COM (READ))
  (SELECTQ (CAR COM)
    (INDUCT (COND ((EQ (CADR COM) 'LIST)
                   (LISTINDUCT (CADDR COM) TH))
                  ((EQ (CADR COM) 'NUM)
                   (NUMINDUCT (CADDR COM) TH))
                  (T (PRINT 'WRONG 'TYPE)
                     (COMMAND))))
    (INDHYP (SETQ NBOC (CADR COM))
            (IF (EQ (CADDR COM) '->)
                (SETQ PAT (CAR IH) REMP (CADDR IH))
                (SETQ PAT (CADDR IH) REMP (CAR IH)))
            (PROVE (USEINDHYP TH) (ADD1 N)))
    (GEN (PROVE (LET ((TH TH) (LSUB (CDR COM)))
                  (IF LSUB
                      (SELF (SUBSTEQUAL (NEXTL LSUB) (NEXTL LSUB) TH)
                            LSUB)
                      TH))
                (ADD1 N)))
    (LOOK (PRINT (EVAL (CADR COM))) (COMMAND))
    (T (PRINT 'UNKNOWN 'COMMAND) (COMMAND))))

(DE REDUCE (THH ;; X Y)
  (SETQ X (RED (CAR THH) NIL)
        Y (RED (CADDR THH) NIL))
  (OR (EQUAL X Y)
      [X '= Y]))
```

```
(DE LISTINDUCT (VAR IH) ; IH = INDUCTION HYPOTHESIS ;
  (PSTAR ['BASE])
  (PROVE (SUBST NIL VAR IH) (ADD1 N))
  (PSTAR ['STEP])
  (PROVE (SUBST ['CONS (GENSYM) VAR] VAR IH) (ADD1 N)))

(DE NUMINDUCT (VAR IH)
  (PSTAR ['BASE])
  (PROVE (SUBST 0 VAR IH) (ADD1 N))
  (PSTAR ['STEP])
  (PROVE (SUBST ['ADD1 VAR] VAR IH) (ADD1 N)))


(DE USEINDHYP (TH) (COND
  ((ATOM TH) (IF (NEQ TH PAT) TH
                 (SETQ NBOC (SUB1 NBOC))
                 (IF (ZEROP NBOC) REMP TH)))
  ((EQUAL TH PAT) (SETQ NBOC (SUB1 NBOC))
                  (IF (ZEROP NBOC) REMP
                      (CONS (USEINDHYP (NEXTL TH))
                            (USEINDHYP TH)))))
  (T (CONS (USEINDHYP (NEXTL TH))
           (USEINDHYP TH)))))


(DE SUBSTEQUAL (X Y TH) (COND
  ((EQUAL Y TH) X)
  ((ATOM TH) TH)
  (T (CONS (SUBSTEQUAL X Y (NEXTL TH))
           (SUBSTEQUAL X Y TH)))))
```

9.7 UN PROGRAMME DE DIALOGUE : AZERTYOP

AZERTYOP est un petit programme de comprehension de scene, d'action sur la scene comprise, de dialogue enfin a son sujet.

Que le lecteur considere AZERTYOP comme un robot ingenu, capable dans les limites de son univers propre, de voir, de saisir, de se souvenir, de parler, d'interroger, de protester (contre les ordres idiots), le tout tres poliment.

Voici une des aventures d'AZERTYOP qu'on appelle a la vie en chantant a VLISP 16 :

(AZERTYOP)

Le partenaire d'AZERTYOP tape ses ordres et questions a la suite du prompteur "?", AZERTYOP se nomme lui-meme a chacune de ses reponses.

```
?(AZERTYOP)
 (AZERTYOP : BJOUR MSIEU)
?(LE CUBE 1 EST PAR TERRE)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(2 EST SUR LUI)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(LE CUBE 3 EST PAR TERRE)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(VOYONS)
       (3 SUR TERRE)
       (2 SUR 1)
       (1 SUR TERRE)
       (DABA)
?(OU EST 3)
 (AZERTYOP : PAR TERRE IL EST MSIEU)
?(PREND LE)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(VOYONS)
       (3 MAIN)
       (2 SUR 1)
       (1 SUR TERRE)
       (DABA)
 ET JE TIENS 3
?(MET LE SUR 2)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(VOYONS)
       (3 SUR 2)
       (2 SUR 1)
       (1 SUR TERRE)
       (DABA)
```

```
?(PREND LE CUBE 4)
 (AZERTYOP : YA PAS DE 4 MSIEU)
?(PREND LE CUBE 2)
 (AZERTYOP : JPEU PAS MSIEU YA 3 DESSUS)
?(4 EST PAR TERRE)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(VOYONS)
        (4 SUR TERRE)
        (3 SUR 2)
        (2 SUR 1)
        (1 SUR TERRE)
        (DABA)
?(MET 3 SUR LUI)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(VOYONS)
        (3 SUR 4)
        (4 SUR TERRE)
        (2 SUR 1)
        (1 SUR TERRE)
        (DABA)
?(MET 1 SUR 1)
 (AZERTYOP : PERSONNE Y PEU FAIRE UNE CHOSE COMME CA MSIEU)
?(PREND 3)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(OU EST IL)
 (AZERTYOP : JELTIEN BIEN MSIEU)
?(PREND 4)
 (AZERTYOP : CAISSE QUEJFAI DE 3 MSIEU ?)
?(POSE LE PAR TERRE)
 (AZERTYOP : OUI MSIEU COMPRIS MSIEU)
?(VOYONS)
        (3 SUR TERRE)
        (4 SUR TERRE)
        (2 SUR 1)
        (1 SUR TERRE)
        (DABA)
?(BYE)
 (AZERTYOP : RVOIR MSIEU)
```

Et voici le listing de AZERTYOP. Dans son etat AZERTYOP est incomplet et inconsistant, le lecteur est cordialement convie a le corriger et l'augmenter de toutes les manieres, apres examen du programme ou il trouvera quelques unes des techniques souvent mises en jeu aujourd'hui dans les programmes de dialogue.

Sur un terminal video, une extension naturelle sera la visualisation de la scene et l'animation des deplacement et saisies d'AZERTYOP qu'on munira pour l'occasion d'une pince.

```
(DE AZERTYOP (;; PHRASE)
  (PRINT '(AZERTYOP : BJOUR MSIEU))
  (SETQ WORD NIL DABA [['DABA]] FOCUS NIL #OBJ NIL #REL NIL #LOC NIL)
  (WHILE (NOT (EQUAL (SETQ PHRASE (READ)) '(BYE)))
         (OR (EVAL NET (GET 'PHRASE 'NET) PHRASE)
             (PRINT '(AZERTYOP : ZAI RIEN COMPRIS MSIEU))))
  '(AZERTYOP : RVOIR MSIEU))

(DE EVAL-NET (NET PHRASE) (COND
  ((NULL NET) NIL)
  ((EVAL-CLAUSE (CAR NET) PHRASE))
  (T (EVAL-NET (CDR NET) PHRASE))))

(DE EVAL-CLAUSE (CLAUSE PHRASE)
  (IF (NULL CLAUSE) (LIST PHRASE)
      (SETQ LASTWORD WORD WORD (CAR PHRASE))
      (IF (ATOM (CAR CLAUSE))
          (IF (EQ (NEXTL CLAUSE) WORD)
              (EVAL-CLAUSE CLAUSE (CDR PHRASE)))
          (SELECTQ (CAAR CLAUSE)
            ($ACT (EPROGN (CDAR CLAUSE))
                  (EVAL-CLAUSE (CDR CLAUSE) PHRASE))
            ($OR (IF (MEMQ WORD (CDAR CLAUSE))
                     (EVAL-CLAUSE (CDR CLAUSE) (CDR PHRASE))))
            ($TEST (IF (EVAL (CADAR CLAUSE))
                       (EVAL-CLAUSE (CDR CLAUSE) (CDR PHRASE))))
            ($CALL (SETQ AUX (EVAL-NET (GET (CADAR CLAUSE) 'NET)
                                       PHRASE))
                   (IF AUX (EVAL-CLAUSE (CDR CLAUSE) (CAR AUX))))
            ()))))

(DF DEF-NET (L) (PUT (CAR L) (CDR L) 'NET))

(DEF-NET PHRASE
 (VOYONS ($ACT (SCENE)))
 (($CALL NG) ($ACT (SETQ #OBJ #NG))
  EST ($CALL LIEU) ($ACT (DECLARATIVE)))
 (PREND ($CALL NG-LE) ($ACT (SETQ #OBJ #NG) (IMPER-1)))
 (($OR MET POSE) ($CALL NG-LE) ($ACT (SETQ #OBJ #NG))
  ($CALL LIEU) ($ACT (IMPER-2)))
 (OU EST ($CALL NG-IL) ($ACT (SETQ #OBJ #NG)(WHERE-Q)))
 (($OR DE DU) ($CALL NG) ($ACT (FOCUS-IT #NG) (P-OUI-MSIEUR)))
 )

(DEF-NET NG
 (($TEST (NUMBP WORD)) ($ACT (SETQ #NG LASTWORD)))
 (LE CUBE ($TEST (NUMBP WORD)) ($ACT (SETQ #NG LASTWORD)))
 )
```

```
(DEF-NET LIEU
 (PAR TERRE ($ACT (SETQ #LOC 'TERRE #REL 'SUR)))
 (SUR ($ACT (SETQ #REL 'SUR)) ($CALL NG-LUI) ($ACT (SETQ #LOC #NG)))
 (SOUS ($ACT (SETQ #REL 'SOUS)) ($CALL NG-LUI) ($ACT (SETQ #LOC #NG)))
 )

(DEF-NET NG-LE
  (($CALL NG))
  (LE ($ACT (SOLVE)))
  )

(DEF-NET NG-IL
  (($CALL NG))
  (IL ($ACT (SOLVE)))
  )

(DEF-NET NG-LUI
  (($CALL NG))
  (LUI ($ACT (SOLVE)))
  )

(DE PRESENT (-P- DABA) (COND
  ((NULL DABA) NIL)
  ((MATCH -P- (NEXTL DABA)))
  (T (PRESENT -P- DABA))))

(DE MATCH (-P- -D-) (COND
  ((AND (NULL -P-) (NULL -D-)) T)
  ((OR (NULL -P-) (NULL -D-)) NIL)
  ((ATOM (CAR -P-)) (IF (EQ (NEXTL -P-) (NEXTL -D-))
                        (MATCH -P- -D-)))
  ((EQ (CAAR -P-) '/,)
   (MATCH (CONS (EVAL (CADAR -P-)) (CDR -P-)) -D-))
  ((EQ (CAAR -P-) '/!)
   (IF (MATCH (CDR -P-) (CDR -D-))
       (SET (CADAR -P-) (CAR -D-))))))

(MCHAR /! (LAMBDA () ['/! (READ)]))
(MCHAR /, (LAMBDA () ['/, (READ)]))


(DE PRINZ L
  (PRINT (APPEND '(AZERTYOP :) L)))

(DE SCENE () (MAPC DABA (LAMBDA (X) (TTAB 5) (PRINT X)))
  (IF (PRESENT '(!X MAIN) DABA) (PRINT 'ET 'JE 'TIENS X)))

(DE SOLVE () (SETQ #NG (NEXTL FOCUS)))
```

```
(DE IN-DABA (X) (SETQ DABA (CONS X DABA)))
(DE OUT-DABA (X) (OUDA X DABA))
(DE OUDA (X DB) (IF (EQUAL X (CAR DB)) (RPLACB DB (CDR DB))
                    (OUDA X (CDR DB))))

(DE P-ABSURDE ()
  (PRINZ 'C/'EST 'SAUF 'VOT 'RESPECT 'MSIEU 'ABSURDE))
(DE P-DE-QUI ()
  (PRINZ 'DE 'QUI 'VOUS 'CAUSEZ 'MSIEU '/?))
(DE P-YAPAS (X)
  (PRINZ 'YA 'PAS 'DE X 'MSIEU))
(DE P-OUI-MSIEU ()
  (PRINZ 'OUI 'MSIEU 'COMPRIS 'MSIEU))

(DE FOCUS-IT (X) (SETQ FOCUS (CONS X FOCUS)))

(DE DECLARATIVE () (COND
  ((EQ #REL 'SOUS) (P-ABSURDE))
  ((OR (NULL #OBJ) (NULL #LOC)) (P-DE-QUI))
  ((DECL DABA))))

(DE DECL (DB) (COND
  ((NULL DB) (IN-DABA [#OBJ 'SUR #LOC]) (FOCUS IT #OBJ) (P-OUI-MSIEU))
  ((MEMQ #OBJ (NEXTL DB)) (PRINZ #OBJ 'EXISTE 'DEJA 'MSIEU))
  (T (DECL DB))))

(DE IMPER-1 () (COND
  ((NULL #OBJ) (P-DE-QUI))
  ((PRESENT '(!X SUR ,#OBJ) DABA)
   (PRINZ 'JPEU 'PAS 'MSIEU 'YA X 'DESSUS) (FOCUS-IT X))
  ((PRESENT '(!X MAIN) DABA) (COND
    ((EQ X #OBJ) (PRINZ 'JELTIEN 'DEJA 'MSIEU) (FOCUS-IT #OBJ))
    (T (PRINZ 'CAISSE 'QUEJFAI 'DE X 'MSIEU '/?) (FOCUS-IT X))))
  ((PRESENT '(,#OBJ SUR !X) DABA)
   (OUT-DABA [#OBJ 'SUR X]) (IN-DABA [#OBJ 'MAIN])
   (FOCUS-IT #OBJ) (P-OUI-MSIEU))
   (T (FOCUS-IT #OBJ) (P-YAPAS #OBJ))))

(DE WHERE-Q ()
  (IF (NULL #OBJ) (P-DE-QUI)
      (FOCUS-IT #OBJ)
      (COND
       ((PRESENT '(,#OBJ MAIN) DABA) (PRINZ 'JELTIEN 'BIEN 'MSIEU))
       ((PRESENT '(,#OBJ SUR !X) DABA)
        (IF (EQ X 'TERRE)
            (PRINZ 'PAR 'TERRE 'IL 'EST 'MSIEU)
            (PRINZ 'IL 'EST 'SUR X 'MSIEU)))
       ((PRESENT '(!X SUR ,#OBJ) DABA)
        (PRINZ X 'EST 'SUR 'LUI 'MAIS #OBJ 'EST 'NULLE 'PART '/,
               'YA 'COMME 'CA 'DES 'OBJETS 'KISONT 'NULLE 'PART))
       (T (P-YAPAS #OBJ)))))
```

```
(DE IMPER-2 () (COND
  ((OR (NULL #OBJ) (NULL #LOC)) (P-DE-QUI))
  ((EQ #OBJ #LOC) (PRINZ 'PERSONNE 'Y 'PEU 'FAIRE 'UNE 'CHOSE 'COMME
        'CA 'MSIEU))
  ((EQ #REL 'SOUS) (P-ABSURDE))
  ((PRESENT '(,#OBJ MAIN) DABA)
   (IF (AND (NEQ #LOC 'TERRE) (PRESENT '(!X SUR ,#LOC) DABA))
       (PRINZ 'JPEUPA 'MSIEU 'YA X 'SUR #LOC)
       (OUT-DABA [#OBJ 'MAIN]) (IN-DABA [#OBJ 'SUR #LOC])
       (FOCUS-IT #OBJ) (P-OUI-MSIEU)))
  ((PRESENT '(!X MAIN) DABA)
   (PRINZ 'CAISSE 'QUE 'JFAIS 'DE X 'MSIEU '/?) (FOCUS-IT X))
  ((PRESENT '(,#OBJ SUR !X) DABA)
   (FOCUS-IT #OBJ)
   (COND
     ((EQ X #LOC) (PRINZ 'ILYEST 'DEJA 'MSIEU))
     ((OR (PRESENT '(!X SUR ,#OBJ) DABA) (PRESENT '(!X SUR ,#LOC) DABA))
      (PRINZ 'JPEUPA 'MSIEU 'YA X 'DESSUS))
     (T (OUT-DABA [#OBJ 'SUR X]) (IN-DABA [#OBJ 'SUR #LOC])
        (P-OUI-MSIEU))))
  (T (P-YAPAS #OBJ))))
```

TABLE D'INDEX DU MANUEL VLISP 16